FILE COPY

(2)

Naval Ocean Research and Development Activity
November 1989 Report 236

Data Base Structure to Support the Production of the Digital Bathymetric Data Base

AD-A227 915

DTIC
ELECTE
OCT 30 1990
S B D

James E. Braud
John L. Breckenridge
James E. Current
Jerry L. Landrum
Mapping, Charting, and Geodesy Division
Ocean Science Directorate

Approved for public release; distribution is unlimited. Naval Ocean Research and Development Activity, Stennis Space Center, Mississippi 39529-5004.

Foreword

This presentation examines the problems encountered in dealing with a tremendous volume of collected bathymetric data that must be processed, edited, and stored for later use in the production of the Digital Bathymetric Data Base and other Navy products. Gridding the data to the minimum resolution that is supported by the collection platform's sensors can greatly reduce the volume of data without a significant impact on data quality. Further reduction of the data storage requirements can be achieved through the use of partial grids and a newly developed compression algorithm that combines run-length encoding and relative encoding techniques. Once the volume of data is reduced to a manageable size, existing data base models can be modified to access partial grids as single entities for storage and retrieval.

[illegible]

W. B. Moseley
Technical Director

[illegible]

J. B. Tupaz, Captain, USN
Commanding Officer

Executive Summary

The Naval Ocean Research and Development Activity was tasked and sponsored by the Naval Oceanographic Office to develop a computer model that would support the storage and retrieval of the 7 billion points of collected bathymetry that are used to support the Ocean Survey Program. These data points are stored at a resolution that is beyond the noise level induced by the collection sensors. Gridding these bathymetric data to an acceptable resolution of 0.1 minute for areas of actual coverage can greatly reduce storage requirements. Further reductions can be achieved through the use of the compression algorithms developed as a result of this project. An overall compression ratio of 500 to 1 was demonstrated using actual data supplied by the Naval Oceanographic Office.

Once the data base is reduced in size to a manageable level, relational data base management systems become practical for organizing information related to the bathymetry in order to provide easy multikeyed access. The Structured Query Language, a de facto standard for data base retrieval, can be used to query the data base to obtain tables of path names for the files that contain the compressed bathymetry. A suggested format for these files and the relational data base structure is presented in this report.

Acknowledgments

The Naval Digital MC&G Analysis Program (DMAP) would like to acknowledge its appreciation for the cooperation and support it has received from the personnel in the Bathymetry Division, Geophysics Department of the Naval Oceanographic Office. In particular, the discussions with Donald Doyle and Richard Sandy provided valuable background material for this report. Also, special recognition is given to Jonathan Bryant of the Naval Oceanographic Office for his efforts in supplying the 12 data sets and graphics (Appendix E) that were used by DMAP in the evaluation.

This study was sponsored by the Naval Oceanographic Office (Program Element 980101) and was conducted as part of the Naval DMAP's Product Optimization efforts.

Accession For	
NTIS GRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Contents

Data Base Structure to Support the Production of the Digital Bathymetric Data Base

I. Introduction

Because of the rapid advances in computer hardware and software, particularly in the increased capabilities of moderately priced graphic workstations and high-density optical storage devices, the Bathymetric Division of the Naval Oceanographic Office (NAVOCEANO) possesses the potential for large gains in productivity and enhanced data quality for the digital bathymetric data products in use by the Navy. In recognition of these developments NAVOCEANO has acquired the Bathymetric Edit Analysis and Matrixing System (BEAMS), which consists of powerful graphic workstations networked to a central minicomputer that provides file server facilities and allows the workstations to share tape drives, disk drives, printers, plotters, etc.

NAVOCEANO is tasked with the collection of bathymetric data in support of the Ocean Survey Program (OSP). These data have been collected over the past 30 years using up to four deep-ocean survey ships. Data from the ships are processed and compressed at NAVOCEANO and are stored in chronological order for each ship as they are collected. This data base currently resides in the form of hundreds of reels of tape. Because of the volume of data (currently over 7 billion points with X,Y,Z values representing longitude, latitude, and depth, respectively), there are no current means of directly accessing the data by area. With the advent of optical disk technology that can provide direct access to billions of bytes of data at a reasonable cost, a computer data model should be developed that will support the multikeyed storage and retrieval of this bathymetric data. The data model should include methods of associating other pertinent information with each X,Y,Z value, such as the name of the collection platform, the date and time that the data were collected, the type and quality of the navigation and sonar used, etc.

II. Objective

This report addresses two main areas of interest to NAVOCEANO in its movement toward the workstation environment that is provided by BEAMS. The first area is data compression. If multikeyed access to the bathymetric data is to be achieved, then much of these data must reside on some type of directly accessible media, such as magnetic or optical disk drives. Because of the enormous size of the bathymetric data base and the current state of technology for disk drives, data compression is likely to be a necessity for many years to come. This report investigates the incremental development of a compression algorithm that may be of use to NAVOCEANO.

The second area deals with the development of a data base structure that will support NAVOCEANO's bathymetric data query needs. Also, a data flow is developed, which shows the various processing stages necessary to convert the existing or newly collected bathymetry into the new data base structure for use in producing the final gridded product.

III. Data Compression

A. Why Data Compression?

The BEAMS system, as well as the industry in general, is moving away from large, centralized, mainframe systems toward a distributed environment dominated by workstations, personal computers, and local area networks (LAN). Although advancements in mass storage technology, such as the availability of high-density Write Once Read Many (WORM) and Compact Disk Read Only Memory (CD-ROM) drives, should ease the on-line bathymetric data storage problem, the vast quantity of uncompressed data still presents considerable burden in the areas of Input/Output (I/O) device speed, LAN and I/O bandwidth, and incorporation into existing data base management systems (DBMS).

Advancements in central processing unit (CPU) speeds have outpaced advancements in I/O bandwidth. The limiting factor in most data base manipulative operations is the access speed of the disk drive and the transfer rate of the data to main memory. In the past, multiuser systems utilized the time waiting for data transfer from I/O devices to memory to process other user's CPU requests. With today's single-user workstations, much of this I/O transfer time results in CPU idle time that could be used to decompress data as it is read in. In some cases this idle time can be minimized by the use of buffers and disk caching; however, these methods, combined with a compression of the data, would further reduce the I/O wait time because it would require far less time for the disk drive to read and transfer compressed data.

In dealing with data base management techniques, an assumption is generally made regarding the storage and retrieval of data. That assumption is that the majority of accesses to the data base will be for data queries as opposed to data updates or inserts. Thus, most DBMSs optimize the data retrieval process at the expense of the data base modification process.

The same assumption can be made when dealing with the compression and decompression of gridded bathymetric data. The compression technique is a two-pass process and is relatively slow in comparison to the decompression. This fact, however, does not present a significant problem, since the compression process is basically a one-time occurrence as opposed to the retrieval process, which may occur many times. The speed of the compression algorithm is made even less significant when it is considered that the prime reason for its slowness is because the enormous size of the uncompressed file results in excessive delays due to I/O wait time. With the reduced size of the compressed files, these delays are much less apparent in the decompression process.

B. Compression Through Gridding

The collected bathymetric data are currently processed by NAVOCEANO by survey operation (SURVOP). The data collected during the approximate 1-month SURVOP are stored on tapes and sent to NAVOCEANO, where the data are filtered, edited, and reformatted into a highly compressed bit-packed form and stored chronologically via tape. Discrete depth values, along with the lateral distance from the ship's position, are stored sequentially with the time at which the data were collected. The ship's heading is also stored, which allows for the computation of the depth position based on the lateral distance's perpendicular orientation to the ship's heading. The navigation data are separately processed and stored. Data retrieval is accomplished by matching the position time with the depth collection time and decompressing appropriate depth information.

During collection, estimated depths are established for discrete points based on the two-way travel time of a pulse of sound (known as a ping) as it is detected by multiple acoustic sensors directed at fixed angles across the beam of the survey platform. These sensors recognize the first significant return of sound as a sample of the shallowest depth found in a large insonified area, or footprint, that results from the geometric spreading of the sound over the distance traveled and the resolution angle of the acoustic sensor (beamwidth). Although the depth position is recorded as the center of the footprint, the actual depth position may occur anywhere within this area. All these errors, combined with positioning error of the ship and variations in the sound velocity profile of the water column, contribute to the noise inherent in the collection system. Spectral analysis studies with historic bathymetric data have revealed that because of this noise level, the accuracy of the depth data within a swath area is insufficient to support a spatial resolution less than an approximate 200-m grid spacing. NAVOCEANO has therefore determined that the 0.1-minute Digital Bathymetric Data Base (DBDB.1) is sufficient for storing the highest resolution bathymetry for areas of swath coverage.

Gridding of the bathymetric data offers tremendous advantages in reducing the storage requirements of the bathymetric data without losing data resolution beyond the threshold of noise that is inherent in the collection system. As an example, the depth data stored in its highly compressed bit-packed format, which was used to produce the 12 data sets provided to NORDA for evaluation, required about 130 Mbytes of storage. The resulting 0.1-minute gridded data sets had a combined storage requirement of only about 3.3 Mbytes. Thus, simply by gridding the existing data, a compression ratio of nearly 40 to 1 can be achieved.

C. Compression Algorithm

For this analysis, NAVOCEANO provided NORDA with nearly 256,000 gridded depths within a geographic 1° × 1° area. These data consist of 12 data sets; each set is represented by a full 600 × 600 array of ASCII depth values.

The overwhelming majority of data values in each data set is zero, with data occurring only for limited swath areas within the total 1° square. Data values are to the 0.1-m vertical resolution, which is beyond the accuracy of current collection systems, but was derived as a result of averaging depth values during the gridding procedure. Although higher compression ratios could be achieved by scaling the vertical resolution to the accuracy of the collected data, the 0.1-m resolution is not degraded throughout the development of the compression algorithm.

The first stage of data compression was simply to process the file into X, Y, and Z components. The nonzero data values were written to an output file with one line per data point using a format of (2I3,I5). The X and Y components, with values ranging from 0 to 600 increments per degree, represent the matrix coordinates of the data point. The depth value is multiplied by 10 and the quantized integer result is written using an I5 format. Although the combined size of the files is reduced slightly for the majority of the 12 data sets, the initial purpose of this step was to convert the data to a format acceptable by the

BEAMS Digital Terrain Modeling (DTM) package. The BEAMS program XYZDRV converts ASCII point data (one point per line) to the DTM .XYZ file format.

The 12 ASCII files containing grid coordinates and depth values were downloaded from the VAX minicomputer to a Zenith-248 in order to simplify program development. The 255,944 data points in these files required 3,071,328 bytes when stored in this format. At this point, the goal of the project is to develop nondegrading compression and decompression algorithms that would significantly reduce the storage requirements of the bathymetric data base. Because of the bit manipulative capabilities and transportability, "C" was chosen as the program development language.

D. Partial Grids

The first step in the compression process of these data points is to eliminate the need to store the X and Y components of the data. This elimination would result in a completely filled array much like the original data, with the exception that instead of the full 1° square area of coverage, only the data within a minimum bounding rectangle would be stored. An initial pass through the data easily determines maximum and minimum values for both X and Y. These maximum and minimum values are stored as header information with the compressed file, such that a decompression routine is able to read the data into a much smaller rectangle or partial grid starting at grid cell (MINX,MINY). The dimensions of the partial grid are defined as MAXX-MINX + 1 cells wide and MAXY-MINY + 1 cells high. Also at this stage, 16-bit binary integer values, which require only 2 bytes for each bathymetric point, are used in place of the 5 bytes required for ASCII representation.

This algorithm requires that the data points be sorted primarily by the Y dimension and secondarily by the X dimension. It was unnecessary to sort the 12 data sets, however, since the NAVOCEANO data provided were originally in the form of an array and were already sorted by column and then row.

The source code for this algorithm is presented in Appendix A. Table 1 shows the compression performance of the algorithm on the 12 data sets. This technique yielded about a 2 to 1 compression ratio overall. In the case of file number 1, the partial grid file was actually larger than the original file because the coverage of the partial grid necessary to include the data in file 1 included about one-half of the entire 1° square (Appendix E).

Although the software developed at this stage was intended to be only an interim step toward developing higher performance compression algorithms, such techniques as the storage of partial grids and the use of binary data representation are carried over into the next phase of algorithm development.

Table 1. Compression of ASCII XYZ files to binary partial grids.

File Number	ASCII XYZ File Size [Bytes]	Partial Grid File Size [Bytes]	Reduction Factor
1	299,726	320,413	0.94
2	252,211	243,419	1.04
3	295,033	73,753	4.00
4	272,582	50,065	5.44
5	268,383	52,007	5.16
6	268,890	76,741	3.50
7	349,295	156,989	2.22
8	376,296	138,793	2.71
9	300,285	182,119	1.65
10	284,750	228,013	1.25
11	310,685	170,135	1.83
12	47,240	8,697	5.43
Total	3,325,376	1,701,144	1.95

E. Run Encoding

The second phase of the compression algorithm development was to reduce the redundancy in the Z coordinate. Much as in the case of images, large shifts in the data values of adjacent pixels or grid cells are rare. The bathymetric files, however, have a high frequency of null cell values, indicating that data do not exist for the cell. These null values are represented by a zero depth for that cell. Also, because of the 0.1-m resolution of the bathymetric grid cells, it is unlikely that long strings of equal Z values will occur frequently. This precludes the use of the typical run-length encoding technique for Z value storage. With run-length encoding a single value, along with its number of times of occurrence, is used to represent a string of values. This technique, however, is useful to represent strings of zero depths that do occur frequently.

Taking advantage on the dependencies that typically exist between successive grid cells, a different type of run encoding was devised for the storage of the Z values. This algorithm outputs an 8-bit value corresponding to the run length, or number, of Z values to follow. This number is then followed by the 16-bit integer representation of the starting Z value. Each successive byte in the run represents a ±7-bit delta from the previous value. A Z value that cannot be represented by a 7-bit delta from the previous value terminates the current run and starts a new run. The exception to this scheme is the occurrence of a null value. This null value is represented by a maximum negative displacement from the preceding Z value. Therefore, the maximum acceptable displacement ranges from -127 to 127 units (in this case a unit is

Table 2. Compression of ASCII XYZ files to run encoded partial grids.

File Number	ASCII XYZ File Size [Bytes]	Partial Grid File Size [Bytes]	Reduction Factor
1	299,726	27,813	10.78
2	252,211	21,130	11.94
3	295,033	23,214	12.71
4	272,582	21,432	12.72
5	268,383	21,158	12.68
6	268,890	21,209	12.68
7	349,295	28,666	12.18
8	376,296	34,539	10.89
9	300,285	24,533	12.24
10	284,750	23,644	12.04
11	310,685	30,262	10.27
12	47,240	4,910	9.62
Total	3,325,376	282,510	11.77

Table 3. Compression of ASCII XYZ files to bit-packed partial grids.

File Number	ASCII XYZ File Size [Bytes]	Partial Grid File Size [Bytes]	Reduction Factor
1	299,726	26,056	11.50
2	252,211	18,940	13.32
3	295,033	20,557	14.35
4	272,582	18,839	14.47
5	268,383	17,602	15.25
6	268,890	16,919	15.89
7	349,295	26,598	13.13
8	376,296	34,796	10.81
9	300,285	19,782	15.18
10	284,750	21,207	13.43
11	310,685	28,537	10.89
12	47,240	4,422	10.68
Total	3,325,376	254,255	13.08

0.1 m), with -128 representing a null value and having no effect on the summation of delta's computation. A null value run length of 4 or greater also terminates the run, and a null run is created. A null run is represented by a zero byte followed by the actual run length.

As indicated by Table 2, this algorithm resulted in a significant compression of the original point data by a factor of about 10 to 1. The source code for the compression routine is listed in Appendix B. One of the problems with this algorithm is the limit placed on the size of the Z value. Sixteen bits allows for a maximum depth of 6553.6 m, which although adequate for this sample data, would not suffice for worldwide coverage. Also, an analysis of the run lengths for nonzero Z values revealed that for several of the data sets, an 8-bit delta value is insufficient to produce run lengths of adequate size. Fragmentation of the runs into average lengths of 4 or 5 Z values causes excessive overhead due to the requirements to store a run-length value and 16-bit Z start value for each run. A higher allowable delta value would have been able to concatenate these runs at the expense of 1 or 2 bits per Z value.

F. Bit Packing

To solve the problems associated with the previous algorithm and to achieve higher compression ratios, the run-encoding compression algorithm was modified to contain an optimized number of bits for the starting Z value associated with each run within a partial grid and for the partial grid's delta Z values. Within partial grids the number of bits used to store these values is fixed and is determined by data analysis during the first pass. The starting Z value for each run is stored as an offset + 1 from the minimum Z value in the data set. The number of bits required is no more than is needed to store this offset for the maximum Z value in the data set. The number of bits needed to store the delta values is determined by choosing a size that will accommodate 95% of the changes in adjacent Z values. The 95 percentile was chosen somewhat arbitrarily through experimentation.

Because the resulting output no longer conformed to word and byte boundaries, generic bit-packing routines were developed to place a bit string of up to 32 bits long into an array at any starting bit offset from the start of the array. The corresponding unpacking routine was also developed and results are shown in Table 3. The source code for the compression algorithms is listed in Appendix C. The source code listing for the decompression algorithm is provided in Appendix D.

The header file listed in Table 4 describes the information included at the beginning of each partial grid. It is intended to be flexible enough to store gridded data of any type and resolution varying from 1 to 65536 increments per degree. The resolution is determined by the value of the integer variable GRIDS, which indicates the number of cells in each row. The integer variable MULT is a scale factor that is applied to the Z values. The value of MULT is itself scaled to 0.001 units, such that a MULT value equal to 100 indicates that all Z values stored are to be multiplied by 0.1 to arrive at their original values. The character variable UNIT can be used to indicate that the gridded Z values are stored at some resolution other than meters, i.e., fathoms or feet.

Table 4. Partial grid header description.

```
struct pheadr{
unsigned int degree; /•(lon + 180)•180 + (lat + 90)•/
unsigned int grids;  /•cells per row and grid•/
unsigned int minx;   /•minimum x cell value for data set•/
unsigned int miny;   /•minimum y cell value for data set•/
unsigned int maxx;   /•maximum x cell value•/
unsigned int maxy;   /•maximum y cell value•/
unsigned int minz;   /•minimum stored z value minus 1 in data set•/
int mult;            /•unit multiplier • 1000•/
unsigned char unit;  /•undefined--set to 0 to indicate meters•/
unsigned char pack;  /•undefined--set to 0 to indicate packed format•/
unsigned char ubit;  /•high 4 bits plus 8 is # of bits to store z•/
                     /•low 4 bits used for size of run length code•/
 };
```

G. Future Compression Enhancements

It is in no way implied that the algorithm presented here represents the final word on compression of the partial grids. Nor is it indicated that a single algorithm should be used for all cases. Therefore, included with the header information for each partial grid is the character-variable PACK, which can be used to indicate the type or version of the decompression routine that is needed to unpack the partial grid. Other methods of compression could be developed for storing nonswath data that is collected in the form of random tracks of center beam depths. These data should still be gridded, but could be represented by a series of graphic line segments followed by the Z values of the cells that are crossed by the line segments.

In determining the boundaries of partial grids, the corners of the bounding rectangle are presently chosen with only a single north-up orientation. In cases of collected swath data at other than north-south or east-west survey tracks, this method creates rectangles that are larger than those actually needed to contain the swath data. The excess area must then be filled with null values and, hence, requires additional storage. An alternate method of partial grid compression could be developed that would consider the angle of orientation of each partial grid that would minimize the fill data required.

Also, consideration of the convergence of longitude at the higher latitudes should be considered in grid spacing. A 0.1-minute grid spacing at 75°N represents a much denser horizontal spacial resolution than a 0.1-minute grid spacing at the equator. Provisions should be made to allow for differing grid spacings for both the X and Y dimensions. This could be achieved by replacing the single integer, GRIDS, in the partial grid header with two separate grid spacing values. Borrowing from the scheme used by the Defense Mapping Agency to represent elevation values for the Digital Terrain Elevation Data, the following table could be adopted:

Zone	Latitude	Spacing Lat/Long
I	0-50° N-S	6 × 6 arc seconds
II	50-70° N-S	6 × 12 arc seconds
III	70-75° N-S	6 × 18 arc seconds
IV	75-80° N-S	6 × 24 arc seconds
V	80-90° N-S	6 × 36 arc seconds

IV. Data Base Structure

A. Bathymetric Data Base Management System

Gridded digital bathymetric data do not conform readily to any of the existing data base models that are in use by off-the-shelf DBMSs. Generally, to incorporate a data set into a DBMS requires a storage overhead of about 2.5 times that of the original data set. Since DBMSs do not have provisions for dealing with compressed data, this multiplication factor would have to be applied to the noncompressed bathymetry and would result in a total storage requirement in the tens of gigabytes. Even if these storage requirements were not the limiting factor, the overhead costs in terms of speed for dealing with the massive number of individual records would render the DBMS useless for most bathymetric applications.

The advantages of using modern-day DBMSs are significant, however, and can provide a simple, standardized user or application program interface to the data. In particular, the relational data base model has gained in popularity over the past years, and DBMSs based on this model are available from many different vendors. Off-the-shelf relational DBMSs have been adapted to run on practically every type of computer system from desktop microcomputers to supercomputers. Data bases conforming to this model therefore offer a greater degree of system independence

Table 5. Relational DBMS bathymetric organization.

TABLES	ATTRIBUTES
PLATFORM	**PLATFORM_ID**
	NAME
	SONAR_TYPE_CONFIG
	NAV_TYPE_CONFIG
SURVOP	**SURVOP_ID**
	PLATFORM_ID (Pointer into PLATFORM table)
	START_DATE
	END_DATE
PARTIAL_GRID	**PG_ID**
	SURVOP_ID (Pointer into SURVOP table)
	GRID_SIZE
	NW_CORNER
	SE_CORNER
	START_DATE
	START_TIME
	NAV_QUALITY
	SONAR_QUALITY
	CLASSIFICATION
	FILE_NAME (Pointer to where actual Z value data is stored)
CELL	**ONE_DEGREE_ID**
	PG_ID

and data exchange capabilities. Relational DBMSs present the user with a uniform and consistent view of the data. They are designed to simplify the process of accessing the data in such a way that data queries, which in the past could be performed by only applications programmers, can now be easily performed by the data user. Also, many relational DBMS vendors have adapted a common set of English-like commands and syntax known as the Structured Query Language (SQL) for accessing the data base. This standardization provides cost savings through reduced training requirements and also provides a common user interface to a multitude of DBMSs. Since the cost of off-the-shelf DBMSs can be amortized over many software packages sold, this cost is generally far less than that of in-house development.

Although it is unlikely that the entire bathymetric data base can be incorporated in a DBMS, an alternative is available. If the gridded bathymetric data were delineated into partial grids in such a way that all attributes pertaining to the depths for that partial grid remain constant, then a relational DBMS could be used to organize a directory of partial grids. Important attributes associated with the depth data include sonar and navigation equipment descriptions, SURVOP identifier, security or access indicators, pointers to archived raw or nongridded bathymetry, navigation and sonar quality, general location (the 1° geographic square in which the partial grid is contained), ship name, etc. These attributes could be organized into a relational data base structure in order to take full advantage of relational DBMS capabilities. Instead of providing actual bathymetric depths, data base queries would result in a list of pointers or path names to the appropriate partial grid files that satisfy the conditions of the query. The partial grids would then have to be accessed and possibly decompressed in order to process the actual bathymetry.

Table 5 contains a possible set of tables and attributes needed to access the bathymetric partial grids through a relational DBMS. By far, the largest table would be PARTIAL_GRID. Estimates of the number of records that would be included in this table range up to 1 million. If the relational DBMS used supported binary as opposed to strictly character representation of the attributes, the size of each record would be 25 bytes. This amount would yield a possible total table storage requirement of 25 Mbytes. If strictly ASCII character representation is allowed, then the size of this table alone could easily exceed 100 Mbytes.

The SURVOP table requires about 1000 records to represent the past 30 years of OSP survey operations. The PLATFORM table requires far fewer records, since there is basically one record for each time the navigation or sonar equipment has been upgraded on a survey ship. The CELL table is actually redundant, since the same information can be obtained by examining the NE_CORNER and SW_CORNER of each of the entries in the PARTIAL_GRID table. The purpose of including a CELL table is to speed the retrieval of data by area. Without this table, access to bathymetric data by area would require a sequential search through each record of the PARTIAL_GRID table. The CELL table requires one record for each record in the PARTIAL_GRID table.

B. Processing

To process the bathymetric data for inclusion into this data base structure, the raw data received from each survey operation would be filtered and edited in much the same way as it is presently. The gridding process would then merge navigation and bathymetry to produce the 0.1-minute grid for the swath-covered areas. During this gridding process two alternate methods for partial grid selection are proposed. Originally it was thought that as the bathymetric data is processed chronologically, each time a 1° line of latitude or longitude is crossed or the navigation or sonar collection quality changes, a new partial grid and the appropriate data base records are created. The navigation and sonar quality is generally determined as a result of the operational mode at the time of data collection. In some cases, as on the fringe of loran coverage or with limited Global Positioning System (GPS) coverage, this navigation mode may change every few minutes. The changes cause a proliferation of very small partial grids and have a detrimental effect on the size and performance of the relational DBMS. A filter could be applied that would help to minimize this effect by ignoring mode changes that take place for only short periods. These short-duration mode changes seem to have little effect on the overall navigation quality.

The alternate method is to collect the set of all data for a SURVOP that exists within a 1° geographic square with a certain navigation or sonar mode of operation. This set of gridded data is then combined into a single partial grid, representing bathymetry of consistent quality for that SURVOP in that geographic area. The effect of this method is to minimize the number of partial grids and associated data base records at the expense of possibly larger and less-related partial grids. The partial grids would be larger, since the bathymetry included is no longer contained by a single swath, but by multiple swaths that may occur anywhere within the 1° square. The minimum bounding rectangle defining the partial grid must include not only all swath data, but also all of the null-filled data gaps that exist between swaths within this rectangle. Perhaps a more detrimental effect of this method is the fact that bathymetry swaths with the same data quality may have been collected within that 1° geographic area over a period that may extend over several days of operation. The storage of the start date and time, as well as detailed pointers to the start of the archival raw data for this partial grid, would be of limited value if time gaps this large could exist within a single partial grid. If it is determined that these attributes are not required to resolve anticipated queries, then they could be eliminated, and this method (or a modified method that considers trade-offs between the size and number of partial grids) could be developed.

Based on the two alternatives discussed, estimates of the number of partial grids that would be generated for each SURVOP range from as few as 40 to as many as 1000. Since as many as 1000 SURVOPs have been done for OSP ships over the past 30 years, up to 1 million partial grids and associated relational DBMS records could result. Although large, this size data base is certainly well within the scope of current DBMS technology.

The output of the gridding program is a set of partial grid files and a set of appropriate data base records containing the attributes of these partial grids. These data base records would likely be in ASCII form in separate files with record formats identical to that of the corresponding DBMS tables. The final step is to update the relational DBMS tables by inserting the newly formed data base records, compressing the partial grids, and copying them to the appropriate directories.

C. Production

At this time, the 0.1-minute bathymetric grid is not intended as a NAVOCEANO product and will not likely be available for distribution outside of NAVOCEANO. It will be used to support the production of large-scale, special-purpose charts required by the Navy. There is also widespread interest in a 0.5-minute grid product in areas with sufficient bathymetric coverage. Such a product would have immediate Antisubmarine Warfare (ASW) applications. In recognition of this application, the Defense Mapping Agency (DMA) has validated a 0.5-minute Digital Bathymetric Data Base (DBDB-0.5) product. The 0.1-minute bathymetric data base described here, in conjunction with interactive graphic workstations, would provide valuable tools for the creation and maintenance of the DBDB-0.5.

The 0.1-minute grid would allow for queries of the data base to be made by geographic area. The resulting list of partial grids should be sorted in reverse order of quality, such that as each partial grid's data values are placed into the workstation's memory, cells of higher-quality data automatically replace cells of lower quality. As the lowest quality data, the DBDB5 (5-minute grid bathymetry) would be interpreted to a 0.5-minute grid to provide an initial overlay. Then any existing 0.5-minute grid data would replace this background in reverse order of quality. Purely interpreted data would have the lowest priority, followed by data that have been graphically edited and regridded. Bathymetry data at the 0.1-minute resolution collected as a result of random ship tracks, SEABEAM, or OSP surveys would also be accessed in reverse order of quality and regridded to 0.5-minute resolution. Each of these data sets would likewise replace any data from previous sets. The end result would be fully populated, 0.5-minute grid representing the best quality data available. Of course, this grid would likely be far from the finished product and would need additional editing. A separate grid or overlay could be simultaneously constructed in a similar manner to represent each cell's associated quality. When the bathymetry grid is contoured for display, this quality overlay could provide color shading to assist the operator in making editing decisions with respect to data quality or security classification.

V. Conclusions

The NAVOCEANO internal bathymetric data base currently consists of a set of 500 reels of highly compressed bathymetric data. Each tape may contain the bathymetry from several survey operations (normally each survey operation is equivalent to about 1 month of survey data). Within each survey operation file, data are stored as they are collected—chronologically. This method creates a serious problem when data are to be accessed by area, since data for a certain area may be located on several tapes, each of which must be searched sequentially to access the appropriated data. Because of the data base's enormous size (about 20 Gbytes), the typical solution of multikeyed access via direct storage devices is very expensive and would likely require optical media arranged in a jukebox device.

Spectral analysis of these bathymetric data, conducted by NAVOCEANO, has revealed that these data are stored at a much higher resolution than is warranted due to the combined noise level induced by the collection sensors. The volume of bathymetric data can be reduced by decreasing the storage resolution to that which can be legitimately supported beyond the collection sensor noise level. This method of storage can be accomplished by gridding the data to a 0.1-minute resolution in the areas of actual swath coverage. Analysis reveals that this step alone can result in a 40 to 1 reduction in the storage requirements. This report detailed the development of a compression routine that achieves a further reduction ratio of about 13-to 1 and proposes a format for the resulting partial grid. The format is flexible enough to allow for different grid resolutions, as well as future enhancements to the compression algorithm.

The combined compression ratios have the potential of reducing the current bathymetric storage requirements by a factor of 500, with minimum loss of actual bathymetric data resolution. This would reduce the current 20 Gbytes of storage requirements to a mere 40 Mbytes, which is well within the range of the capacity of inexpensive direct access devices. Data access methods to this data base could also be greatly enhanced by a relational DBMS used to organize information associated with the compressed partial grids. Conditional multikeyed data retrieval could be accomplished both interactively or within procedural language programs through the use of the de facto standard SQL.

VI. Bibliography

1. Held, Gilbert. *Data Compression, Techniques and Applications, Hardware and Software,* 2nd ed., John Wiley & Sons Ltd., 1983,1987.

2. Fox, Chistopher G. *Description, Analysis, and Prediction of Sea-Floor Roughness Using Spectral Models,* Naval Oceanographic Office, Stennis Space Center, MS, TR 279, 1985.

Appendix A

Source Code for Compression to Simple Partial Grid File

```
/*      This routine opens the file that is specified by the first argument
and reads through it to determine the boundaries of the partial grid.
The partial grid is defined as a 2 dimensional array with the first element
containing the value that is located at (minx,miny). The array extends in
the x direction until maxx-minx+1. The y dimension is ymax-ymin+1.
        The first pass through the input file determines the minimum and
maximum values for x and y defined by minx, maxx, miny, and maxy. The second
pass through the input file builds a rectangular array or partial grid
containing the z values. This array is written to the output file that is
defined by the second argument. If there is no z value for a particular
coordinate, a null or zero z value is output.
*/

#include <stdio.h>
#include <fcntl.h>

/*The following structure is used as the header for the output partial grid
file. Not all information is relavent at this time*/

struct pheadr{
        unsigned int degree;/*(lon+180)*180+(lat+90)*/
        unsigned int grids; /*cells per row and grid*/
        unsigned int minx;  /*minimum x cell value for data set*/
        unsigned int miny;  /*minimum y cell value for data set*/
        unsigned int maxx;  /*maximum x cell value*/
        unsigned int maxy;  /*maximum y cell value*/
        unsigned int minz;  /*minimum stored z value minus 1 in data set*/
        int mult;           /*unit multiplier * 1000*/
        unsigned char unit; /*undefined--set to 0 to indicate meters*/
        unsigned char pack; /*undefined--set to 0 to indicate packed format*/
        unsigned char ubit; /*high 4 bits plus 8 is # of bits to store z*/
                            /*low 4 bits used for size of run length code*/

        };

main(argc,argv)
int argc;
char *argv[];

{
struct pheadr head;
char start=1;
int zero=0;
unsigned int x,y,z,minx,maxx,miny,maxy,rlen,fd;
unsigned int current,next,diff;
FILE *fp,*fopen();
```

```
/*check for too few arguments*/

if(argc < 3){
        fprintf(stderr,"Usage: pgrid input_filename output_filename\n");
        exit(1);
        }

/*does the input file exist and can it be opened*/

if((fp=fopen(*++argv, "r"))==NULL){
        fprintf("pgrid: can't open %s\n",*argv);
        exit(1);
        }

/*The input file contains header information such as navigation and sonar
qualitv, latitude and logitude degrees, etc. The format of this header is
subject to change, therefore the routine to read this information has been
separated from this modual.*/

rheader(fp,&head);

/*The reader routine is separated from the main routine to limit the impact
of future format changes. The reader is p.  ided with the file pointer, fp.
It reads an input file record and returns the values of x, y and z.*/

while (reader(fp,&x,&y,&z)){
        if(start){
                /*1st time through--initialize maxx,maxy,miny,minx*/
                minx=maxx=x;
                miny=maxy=y;
                start=0;
                }
        else{
                if(x < minx)
                        minx=x;
                else
                        if(x > maxx)
                                maxx=x;
                if(y < miny)
                        miny=y;
                else
                        if(y > maxy)
                                maxy=y;
                }
        }

/*End of first pass--partial grid dimensions have been defined*/

maxx++;
```

```
maxy++;
rlen=maxx-minx;

/*Go to the beginning of input file for second pass*/

fclose(fp);
fp=fopen(*argv,"r");

/*Create and open output file. If it already exists it will be overwritten.*/

fd=creat(*++argv,0766);
close (fd);

/*Output file must be binary. The next line is non-standard used for TURBO C*/

_fmode=O_BINARY;

fd=open(*argv,2);

/*Read header information*/

rheader(fp,&head);

/*Form output header for partial grid*/

head.minx=minx;
head.miny=miny;
head.maxx=maxx;
head.maxy=maxy;
write(fd,&head,sizeof(head));

/*current is used to indicate how many null values to fill between actual z
values*/

current=0;

/*Read through input file and output z values to partial grid*/
/*reader returns a false or zero value for end of file*/

while(reader(fp,&x,&y,&z)){
        x-=minx;
        y-=miny;

        /*Determine how many null values between this z value and the last*/

        next=y*rlen+x;
        diff=next-current;
        current=next;

        /*Fill partial grid with null z values*/
        while(diff-- > 1)
                write(fd,zero,2);

        /*Write out z value*/

        write(fd,z,2);
        }
/*No need to fill from last z value to end of array--null values are assumed*/
}


/*The following routine is used to read the header information from an xyz
file. The latitude and longitude in degrees is the only information in the
header that is actually read at this time. The variable degree is used to
combine lat and lon into a single integer value. The variable mult=100
indicates that the z values stored in the partial grid have been multiplied
by 10. The variable grids=600 indicates that the resolution in the x and y
dimensions is .1 minutes.*/

rheader(fp,head)
FILE *fp;
struct pheadr *head;
{
char buf[80],i=0;
int lat,lon;
while ((buf[i++]=getc(fp)) != '\n');
sscanf(buf,"%d %d",&lat,&lon);
head->degree=(lon+180)*180+(lat+90);
head->mult=100;
head->grids=600;
}

/*The reader routine reads the xyz file that was written out using a FORTRAN
(2i3,i5) format. The x and y coordinates represent .1 minute offsets from the
corner of the 1-degree cell. The z values are in meters*10. Each xyz value is
written 1 to a line. A zero or false value is returned for end of file.*/

reader(fp,x,y,z)
FILE *fp;
unsigned int *x,*y,*z;
{
char buf[80],j;

/*Check for EOF*/

if((buf[0]=getc(fp)) != EOF){
        j=1;

        /*Get a line until line feed or EOF*/
```

```
          while ((buf[j]=getc(fp)) != '\n' && buf[j++] != EOF);

          /*Fill leading blanks with leading zeros.*/

          for(j=0;j<=4;j++)
                    if(buf[j]==' ')buf[j]='0';

          /*Read from memory using format.*/

          sscanf(buf,"%3d %3d %d",x,y,z);
          return 1;
          }
     else
          return 0;
     }
```

Appendix B

Source Code for Modified Run-encoded Partial Grid Compression

```
/*      This routine opens the file that is specified by the first argument
and reads through it to determine the boundaries of the partial grid.
The partial grid is defined as a 2 dimensional array with the first element
containing the value that is located at (minx,miny). The array extends in
the x direction until maxx-minx+1. The y dimension is ymax-ymin+1.
        The first pass through the input file determines the minimum and
maximum values for x and y defined by minx, maxx, miny, and maxy. The second
pass through the input file builds a rectangular array or partial grid
containing the z values. This array is written to the output file that is
defined by the second argument given in the MS-DOS command line. If there is
no z value for a particular coordinate, a null or zero z value is output.
*/

/*The following structure is used as the header for the output partial grid
file. Not all information contained in the header is relavent at this time*/

struct pheadr{
        unsigned int degree;/*(lon+180)*180+(lat+90)*/
        unsigned int grids; /*cells per row and grid*/
        unsigned int minx;  /*minimum x cell value for data set*/
        unsigned int miny;  /*minimum y cell value for data set*/
        unsigned int maxx;  /*maximum x cell value*/
        unsigned int maxy;  /*maximum y cell value*/
        unsigned int minz;  /*minimum stored z value minus 1 in data set*/
        int mult;           /*unit multiplier * 1000*/
        unsigned char unit; /*undefined--set to 0 to indicate meters*/
        unsigned char pack; /*undefined--set to 0 to indicate packed format*/
        unsigned char ubit; /*high 4 bits plus 8 is # of bits to store z*/
                            /*low 4 bits used for size of run length code*/

        };

#include <stdio.h>

/*The following header file is included in order to permit binary I/O. It is
language specific to the TURBO C compiler*/

#include <fcntl.h>

unsigned int kount,z,last_z;
int fd;    /*fd is the file descriptor for the output file*/
long next,current,diff;
unsigned char buf[300];
unsigned char start=1;

main(argc,argv)
```

```
int argc;
char *argv[];
{
int xc,yc,i,doff;
unsigned int x,y,minx,maxx,miny,maxy,rlen;
unsigned int xsave,ysave,dsave,lat,lon,grids;
FILE *fp,*fopen();
struct pheadr head;

/*check for too few arguments*/

if(argc < 3){
        fprintf(stderr,"Usage: pgrid input_filename output_filename\n");
        exit(1);
        }

/*does the input file exist and can it be opened*/

if((fp=fopen(*++argv, "r"))==NULL){
        fprintf("pgrid: can't open %s\n",*argv);
        exit(1);
        }

/*The input file contains header information such as navigation and sonar
quality, latitude and longitude degrees, etc. The format of this header is
subject to change, therefore the routine to read this information has been
separated from this modual.*/

rheader(fp,&head);

/*The reader routine is separated from the main routine to limit the impact
of future format changes. The reader is provided with the file pointer, fp.
It reads an input file record and returns the values of x, y and z.*/

while (reader(fp,&x,&y,&z)){
        if(start){
                /*1st time through--initialize maxx,maxy,miny,minx*/
                minx=maxx=x;
                miny=maxy=y;
                start=0;
                }
        else{
                if(x < minx)
                        minx=x;
                else
                        if(x > maxx)
                                maxx=x;
                if(y < miny)
                        miny=y;
                else
```

```
                                        if(y > maxy)
                                                maxy=y;
                          }
                }

        /*End of first pass--partial grid dimensions have been defined*/

        maxx++;
        maxy++;
        rlen=maxx-minx;

        /*Go to the beginning of input file for second pass*/

        fclose(fp);
        fp=fopen(*argv,"r");

        /*Create and open output file. If it already exists it will be overwritten.*/

        fd=creat(*++argv,0766);
        close (fd);

        /*Output file must be binary. The next line is non-standard used for TURBO C*/

        _fmode=O_BINARY;

        fd=open(*argv,2);

        /*Read header information*/

        rheader(fp,&head);

        /*Form output header for partial grid*/

        head.minx=minx;
        head.miny=miny;
        head.maxx=maxx;
        head.maxy=maxy;
        write(fd,&head,sizeof(head));
        start=1;

        /*current is used to indicate how many null values to fill between actual z
        values*/

        current=0;

        /*Read through input file and output z values to partial grid*/
        /*reader returns a false or zero value for end of file*/

        while(reader(fp,&x,&y,&z)){
                x-=minx;
        y-=miny;
        next=(long)y*rlen+x;
        diff=next-current;
        if (start){
                /*Output any null runs as required and initialize for 1st
                run*/
                newrec();
                start=0;
                }
        else
                /*If the number of nulls is less than 4 may be more economical
                to include them in z run as -128 values--indicates nulls*/
                if(diff < 4){
                        /*If overflow of z run--start new run*/
                        if(diff+kount > 255)newrec();
                        else{
                                /*Check for offset overflow*/
                                if(kount == 255 || abs(doff=z-last_z)>127){
                                        /*Current run ends here*/
                                        /*If only 1 null value and it can
    fit*/
                                        /*it is more economical to output
    here*/
                                        if(diff==1 && kount<255){
                                                diff--;
                                                buf[kount++ + 2]=-128;
                                                }
                                        /*output this run and start next*/
                                        newrec();
                                        }
                                else{
                                        while(diff>0){
                                                diff--;
                                                buf[kount++ + 2]=-128;
                                                }
                                        buf[kount++ + 2]=doff;
                                        last_z=z;
                                        current=next+1;
                                        }
                                }
                        }
                /*write out current run and null run and start new run*/
                else newrec();
        }

    /*force out current run*/
    diff=0;
    newrec();
    }
```

```
/*The following routine forces out a z-run that is stored in buffif it is not
the first call--start not equal to 0. It recalls putz to output any null
runs that are required. Finally it initializes for a new z run.*/

newrec()
{
if(!start){

        /*kount is the length of the run. Store it in the 1st byte.*/

        buf[0]=kount;
        write(fd,&buf,kount+2);
        }

/*Output any null runs necessary*/

if(diff > 0)
        putz();

/*Store current z value into run buffer.*/

buf[2]=z & 255;
buf[1]=z >> 8;

/*Minimum non-null run legnth is 1 z value long*/

kount=1;

/*last_z is the last z value encountered. Used to calculate the offset of
current
value from the previous z value*/

last_z=z;

/*current is the position in the partial grid that the next z value should be
at if there are no zero fills needed*/

current=next+1;
}


/*PUTZ is a routine that outputs a run of null z values. A run of null values
is defined as a count field followed by a z value of zero. The count field
indicates that the following zero value is to be repeated count + 1 times*/

putz()
{
buf[0]=0;
/*There may be multiple runs necessary to define total string of nulls*/
while(diff){

        /*Output maximum length or whatever is left*/

        /*The 1st byte of the run is length-1. Length ranges from 1 to 256*/

        buf[1]=(diff >= 256) ? 255 : diff-1;

        /*Update the count of nulls remaining*/

        diff-=buf[1];
        diff--;

        /*Force out this run*/

        write(fd,&buf,2);
        }
}


/*The following routine is used to read the header information from an xyz
file. The latitude and longitude in degrees is the only information in the
header that is actually read at this time. The variable degree is used to
combine lat and lon into a single integer value. The variable mult=100
indicates that the z values stored in the partial grid have been multiplied
by 10. The variable grids=600 indicates that the resolution in the x and y
dimensions is .1 minutes.*/

rheader(fp,head)
FILE *fp;
struct pheadr *head;
{
char buf[80],i=0;
int lat,lon;
while ((buf[i++]=getc(fp)) != '\n');
sscanf(buf,"%d %d",&lat,&lon);
head->degree=(lon+180)*180+(lat+90);
head->mult=100;
head->grids=600;
}

/*The reader routine reads the xyz file that was written out using a FORTRAN
(2i3,i5) format. The x and y coordinates represent .1 minute offsets from the
corner of the 1-degree cell. The z values are in meters*10. Each xyz value is
written 1 to a line. A zero or false value is returned for end of file.*/

reader(fp,x,y,z)
FILE *fp;
unsigned int *x,*y,*z;
```

```
{
char buf[80],j;

/*Check for EOF*/

if((buf[0]=getc(fp)) != EOF){
        j=1;

        /*Get a line until line feed or EOF*/

        while ((buf[j]=getc(fp)) != '\n' && buf[j++] != EOF);

        /*Fill leading blanks with leading zeros.*/

        for(j=0;j<=4;j++)
                if(buf[j]==' ')buf[j]='0';

        /*Read from memory using format.*/

        sscanf(buf,"%3d%3d%d",x,y,z);
        return 1;
        }
else
        return 0;
}
```

Appendix C

Source Code for Bit-packed Run-encoded Partial Grid Compression

```
/*      This routine opens the file that is specified by the first argument
and reads through it to determine the boundaries of the partial grid.
The partial grid is defined as a 2 dimensional array with the first element
containing the value that is located at (minx,miny). The array extends in
the x direction until maxx-minx+1. The y dimension is ymax-ymin+1.
        The first pass through the input file determines the minimum and
maximum values for x and y defined by minx, maxx, miny, and maxy. The second
pass through the input file builds a rectangular array or partial grid
containing the z values. This array is written to the output file that is
defined by the second argument given in the MS-DOS command line. If there is
no z value for a particular coordinate, a null or zero z value is output.
*/

/*The following structure is used as the header for the output partial grid
file. Not all information contained in the header is relavent at this time*/

struct pheadr{
        unsigned int degree;/*(lon+360)*180+(lat+90)*/
        unsigned int grids; /*cells per row and grid*/
        unsigned int minx;  /*minimum x cell value for data set*/
        unsigned int miny;  /*minimum y cell value for data set*/
        unsigned int maxx;  /*maximum x cell value*/
        unsigned int maxy;  /*maximum y cell value*/
        unsigned int minz;  /*minimum stored z value minus 1 in data set*/
        int mult;           /*unit multiplier * 1000*/
        unsigned char unit; /*undefined--set to 0 to indicate meters*/
        unsigned char pack; /*undefined--set to 0 to indicate packed format*/
        unsigned char ubit; /*high 4 bits plus 8 is # of bits to store z*/
                            /*low 4 bits used for # of bits for z value
offset*/
        };

#include <stdio.h>

/*The following header file is included in order to permit binary I/O. It is
language specific to the TURBO C compiler*/

#include <fcntl.h>

unsigned int kcunt,z,last_z;
int fd;    /*fd is the file descriptor for the output file*/
long next,current,diff;
unsigned char buf[3000];
unsigned char start=1;
unsigned long mxval,hibit,doff;
unsigned int dbits,dbits,hbits,mhval,bsave,bcnt;

main(argc,argv)
int argc;
char *argv[];
{
int xc,yc,i;
unsigned int x,y,minx,maxx,miny,maxy,rlen,min,max;
unsigned int xsave,ysave,zsave,lat,lon,grids;
float sum1,c1=0;
long cut[]={1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,2<<14,2<<15,
        2<<16,2<<17,2<<18,2<<19,2<<20,2<<21,2<<22,2<<23,2<<24};
unsigned int cnt[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
FILE *fp,*fopen();
struct pheadr head;

/*check for too few arguments*/

if(argc < 3){
        fprintf(stderr,"Usage: pgrid input_filename output_filename\n");
        exit(1);
        }

/*does the input file exist and can it be opened*/

if((fp=fopen(*++argv, "r"))==NULL){
        fprintf("pgrid: can't open %s\n",*arg[illegible] ,
        exit(1);
        }

/*The input file contains header information such as navigation and sonar
quality, latitude and longitude degrees, [illegible] [illegible] format of this header is
subject to change, therefore the routine to read this information has been
separated from this modual.*/

rheader(fp,&head);

/*The reader routine is separated from the main routine to limit the impact
of future format changes. The reader is provided with the file pointer, fp.
It reads an input file record and returns the values of x, y and z.*/

while (reader(fp,&x,&y,&z)){
        if(start){
                /*1st time through--initialize maxx,maxy,miny,minx*/
                minx=maxx=x;
                miny=maxy=y;
                min=max=z;
                start=0;
                }
        else{
                if(x==++xsave && y==ysave){
                        diff=abs(z-zsave);
                        c1++;
                        i=0;
                        while(diff>cut[i++]);
                        cnt[i]+=1;
                        }
                if(x < minx)
                        minx=x;
                else
                        if(x > maxx)
                                maxx=x;
                if(y < miny)
                        miny=y;
                else
                        if(y > maxy)
                                maxy=y;
                if(z < min)
                        min=z;
                else
                        if(z > max)
                                max=z;
                }
        zsave=z;
        xsave=x;
        ysave=y;
        }

/*End of first pass--partial grid dimensions have been defined*/
maxx++;
maxy++;
/*Calculate the value of cbits, the # of bits used to store the offset from
the previous z value. It is calculated as the bits needed to store 95% of
the offsets between consecutive z values.*/

c1*=.95;
sum1=0.;
cbits=0;
while((sum1+=cnt[cbits++])<c1);
cbits--;

/*hbits and mhval are constants at this time. hbits = 8 and mhval = 255 These
variables are used to define the maximum run length and could be experimented
with to possibly further optimize the compression.*/

hbits=8;
mhval=(1<<(hbits))-1;

/*mxval indicates the maximum offset that is allowed for the z value*/
/*if cbits=8, mxval=127--allows an offset of + or - 127*/
```

```
mxval=(1<<(cbits-1))-1;
/*hibit is used to indicate a null value in a z-run. Set to mxval+1*/
hibit=mxval+1;
/*min will be subtracted from all z values. Decrement min to avoid zero*/
min--;
rlen=maxx-minx;
/*calculate the number of bits neededed to store the maximum z value*/
dbits=0;
z=max-min;
while(cut[dbits++]<z);
--dbits;

/*Go to the beginning of input file for second pass*/

fclose(fp);
fp=fopen(*argv,"r");

/*Creat and open output file. If it already exists it will be overwritten.*/

fd=creat(*++argv,0766);
close (fd);

/*Output file must be binary. The next line is non-standard used for TURBO C*/

_fmode=O_BINARY;

fd=open(*argv,2);

/*Read header information*/

rheader(fp,&head);

/*Form output header for partial grid*/

head.minx=minx;
head.miny=miny;
head.maxx=maxx;
head.maxy=maxy;
head.minz=min;
head.pack=head.unit=0;
head.ubit=(dbits-8)<<4 | cbits;
write(fd,&head,sizeof(head));
start=1;

/*current is used to indicate how many null values to fill between actual z
values*/

current=0;

/*Read through input file and output z values to partial grid*/
```

```
/*reader returns a false or zero value for end of file*/

while(reader(fp,&x,&y,&z)){

        x-=minx;
        y-=miny;
        z-=min;
        next=(long)y*rlen+x;
        diff=next-current;
        if (start){
                /*Output any null runs as required and initialize for 1st run*/
                newrec();
                start=0;
                }
        else
                /*If the number of nulls is less than 4 may be more economical
                to include them in z run as -128 values--indicates nulls*/
                if(diff < 4){
                        /*If overflow of z run--start new run*/
                        if(diff+kount > mhval)newrec();
                        else{
                                /*Check for offset overflow*/
                                if(kount == mhval || abs(doff=z-last_z)>mxval){
                                        /*Current run ends here*/
                                        /*If only 1 null value and it can fit*/
                                        /*it is more economical to output here*/
                                        if(diff==1 && kount<mhval){
                                                diff--;
                                                bit_pack(buf,bcnt,cbits,hibit);
                                                bcnt+=cbits;
                                                kount++;
                                                }
                                        /*output this run and start next*/
                                        newrec();
                                        }
                                else{
                                        while(diff>0){
                                                diff--;
                                                /*hibit is used to indicate a*/
                                                /*null value in a z run*/
                                                bit_pack(buf,bcnt,cbits,hibit);
                                                bcnt+=cbits;
                                                kount++;
                                                }
                                        bit_pack(buf,bcnt,cbits,doff);
                                        last_z=z;
                                        bcnt+=cbits;
                                        current=next+1;
                                        kount++;
                                        }
                                        }
                                }
                /*Force out current run and null run. Start new run*/
                else newrec();
        }

/*Force out current run*/

diff=0;
newrec();
while(bsave){
        write(fd,buf,1);
        bsave-=(bsave<8)?bsave:8;
        }
}

/*The following routine forces out a z-run that is stored in buf if it is not
the first call--start not equal to 0. It the calls putz to output any null
runs that are required. Finally it initializes for a new z run.*/

newrec()
{
int knt;
/*Force out current z-run if this is not the first call*/

if(!start){
/*kount is the length of this run. Place it in the buffer*/
        bit_pack(buf,bsave,hbits,kount);
/*knt is the number of bytes to write. If the last byte is not full it is
placed in position buf[0] and the buffer is filled from the last bit used*/
        knt=bcnt >> 3;
        write(fd,&buf,knt);
        buf[0]=buf[knt];
/*bsave is the start bit location of next run*/
        bsave=bcnt % 8;
        }

/*Output null run if needed*/

if(diff > 0)
        putz();

/*Initialize buffer for new z-run*/

bcnt=bsave+hbits;

/*Store current z value into run buffer.*/

bit_pack(buf,bcnt,dbits,z);
bcnt+=dbits;
```

```
/*Minimum non-null run legnth is 1 z value long*/

kount=1;

/*last_z is the last z value encountered. Used to calculate the offset of
current
value from the previous z value*/

last_z=z;

/*current is the position in the partial grid that the next z value should be
at if there are no zero fills needed*/

current=next+1;
}

/*PUTZ is a routine that outputs a run of null z values. A run of null values
is defined as a count field followed by a z value of zero. The count field
indicates that the following zero value is to be repeated count + 1 times*/

putz()
{
long j;
int i;
/*i = # of bits needed to represent a null run. 2 times hbits is currently 16.*/
i=hbits<<1;
while(diff){
        /*output max length null runs (mhval=255) until one is < mhval*/
        j=(diff > mhval) ? mhval : diff-1;
        /*upper half of i bit length of j is =0. Pack into buffer*/
        bit_pack(buf,bsave,i,j++);
        /*update counters*/
        diff-=j;
        bsave+=i;
        bcnt+=i;
        }
}


/*The following routine is used to read the header information from an xyz
file. The latitude and longitude in degrees is the only information in the
header that is actually read at this time. The variable degree is used to
combine lat and lon into a single integer value. The variable mult=100
indicates that the z values stored in the partial grid have been multiplied
by 10. The variable grids=600 indicates that the resolution in the x and y
dimensions is .1 minutes.*/

rheader(fp,head)
FILE *fp;
```

```
struct pheadr *head;
{
char buf[80],i=0;
int lat,lon;
while ((buf[i++]=getc(fp)) != '\n');
sscanf(buf,"%d %d",&lat,&lon);
head->degree=(lon+180)*180+(lat+90);
head->mult=100;
head->grids=600;
}

/*The reader routine reads the xyz file that was written out using a FORTRAN
(2i3,i5) format. The x and y coordinates represent .1 minute offsets from the
corner of the 1-degree cell. The z values are in meters*10. Each xyz value is
written 1 to a line. A zero or false value is returned for end of file.*/

reader(fp,x,y,z)
FILE *fp;
unsigned int *x,*y,*z;
{
char buf[80],j;

/*Check for EOF*/

if((buf[0]=getc(fp)) != EOF){
        j=1;

        /*Get a line until line feed or EOF*/

        while ((buf[j]=getc(fp)) != '\n' && buf[j++] != EOF);

        /*Fill leading blanks with leading zeros.*/

        for(j=0;j<=4;j++)
                if(buf[j]==' ')buf[j]='0';

        /*Read from memory using format.*/

        sscanf(buf,"%3d%3d%d",x,y,z);
        return 1;
        }
else
        return 0;
}


bit_pack(buf,str,n,num)
long num;
unsigned char *buf;
unsigned int str,n;
{
static char msk[8]={0,128,192,224,240,248,252,254};
char *sbyt,*ebyt;
int sbit,ebit,i;
i=str+n;
sbyt=(str >> 3) + buf;
ebyt=(i >> 3) + buf;
sbit=str % 8;
ebit=i % 8;
i=ebyt-sbyt-1;
if(sbyt == ebyt){
        *sbyt &= msk[sbit] | ~msk[ebit];
        *sbyt |= (num << (8-ebit)) & (~msk[sbit] & msk[ebit]);
        }
else{
        *sbyt &= msk[sbit];
        *sbyt++ |= (num >> (n-(8-sbit))) & ~msk[sbit];
        while(i--){
                *sbyt &= 0;
                *sbyt++ |= (num >> ((i << 3) + ebit)) & 255;
                }
        *sbyt &= ~msk[ebit];
        *sbyt |= (num << (8-ebit));
        }
}
```

Appendix D

Source Code for Bit-packed Run-encoded Partial Grid Decompression

```
/*The following program is used to unpack or decompress the gridded data that
was compressed using the bit packed relative run encoding technique. The name of
the input file is given as an argument. The output is to the standard output
device in the form of 1 line per grid cell with the X and Y component formatted
as 2I3 followed by the unscaled Z component as I5.*/

#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <fpack.h>
main(argc,argv)
int argc;
char *argv[];
{
struct pheadr h;
unsigned int fd,x,y,dsave,min,max,miny,minx,maxx,maxy,depth;
_fmode=O_BINARY;
if((fd=open(*++argv,0))<0){
        fprintf(stderr,"pgrid: Could not open %s\n",*argv);
        exit (1);
        }
printf("\n");
read(fd,&h,sizeof(h));
dsave=h.minz;
min=65535;
max=0;
for(y=h.miny;y<h.maxy;y++)
        for(x=h.minx;x<h.maxx;x++)
                if(pread(fd,&depth,&h)>0){
                        if(depth>0){
                                depth+=dsave;
                                printf("%3d %3d %5d\n",x,y,depth);

                                /*The following computation of maximum and
                                minimum depth is unnecessary but is used as
                                a data check*/

                                if(depth < min)
                                        min=depth;
                                else
                                        if(depth > max)
                                                max=depth;
                                }
                        }
                 else{
                        printf("%d %d\n",min,max);
```

```
                              exit(0);
                              }
}

/*The following routine is called for each cell value to be decompressed.*/

pread(fd,depth,head)
int fd,*depth;
struct pheadr *head;
{
static unsigned char start=1;
static unsigned int dbits,cbits,hibit,hbits;
int d;
static int dsave=0,df=0,zf=0;
if(start){
        /*Initially set up values as defined in the header*/
        start=0;
        dbits=(head->ubit>>4)+8;
        cbits=head->ubit&15;
        hbits=8;
        hibit=1<<(cbits-1);
        }
/*zf is the number of zeros in this run*/
if(zf){
        zf--;
        *depth=0;
        return 1;
        }
else
        /*df is the number of depths in this run*/
        if(df){
                d=chk(fd,cbits);
                df--;
                if(d==hibit){
                        *depth=0;
                        return 1;
                        }
                else
                        if(d & hibit)
                                d-=(hibit<<1);
                *depth=dsave+d;
                dsave+=d;
                return 1;
                }
        else{
                d=chk(fd,hbits);
                if(d<0)
                        return 0;
                else
                        if(d){
```

```
                                                dsave=chk(fd,dbits);
                                                *depth=dsave;
                                                df=d-1;
                                                return 1;
                                                }
                                   else{
                                                zf=chk(fd,hbits);
                                                *depth=0;
                                                return 1;
                                                }
                 }
}

/*The following routine reads the compressed file and returns a value
represented by the next n bits*/

long chk(fd,n)
unsigned int fd,n;
{
static unsigned char buf[280];
long num;
static unsigned char *ebyt,*sbyt;
static int lbyt=0,bcnt=0;
int i;
ebyt=((bcnt+n-1)>>3)+buf;
if(ebyt >= lbyt+buf){
         sbyt=(bcnt>>3)+buf
         i=0;
         while(sbyt<([illegible] +buf))
                  buf[i++]=*sbyt++;
         bcnt %= 8
         lbyt=read(fd,&buf[i],256) + i;
         if(lbyt<=i)return -1;
         }
num = unpack(buf,bcnt,n);
bcnt+=n;
return num;
}


unsigned long unpack(buf,str,n)
unsigned char *buf;
unsigned int str,n;
{
static char msk[8]={0,128,192,224,240,248,252,254};
unsigned long num;
unsigned char *sbyt,*ebyt;
int sbit,ebit,i;
i=str+n;
sbyt=(str >> 3) + buf;
```

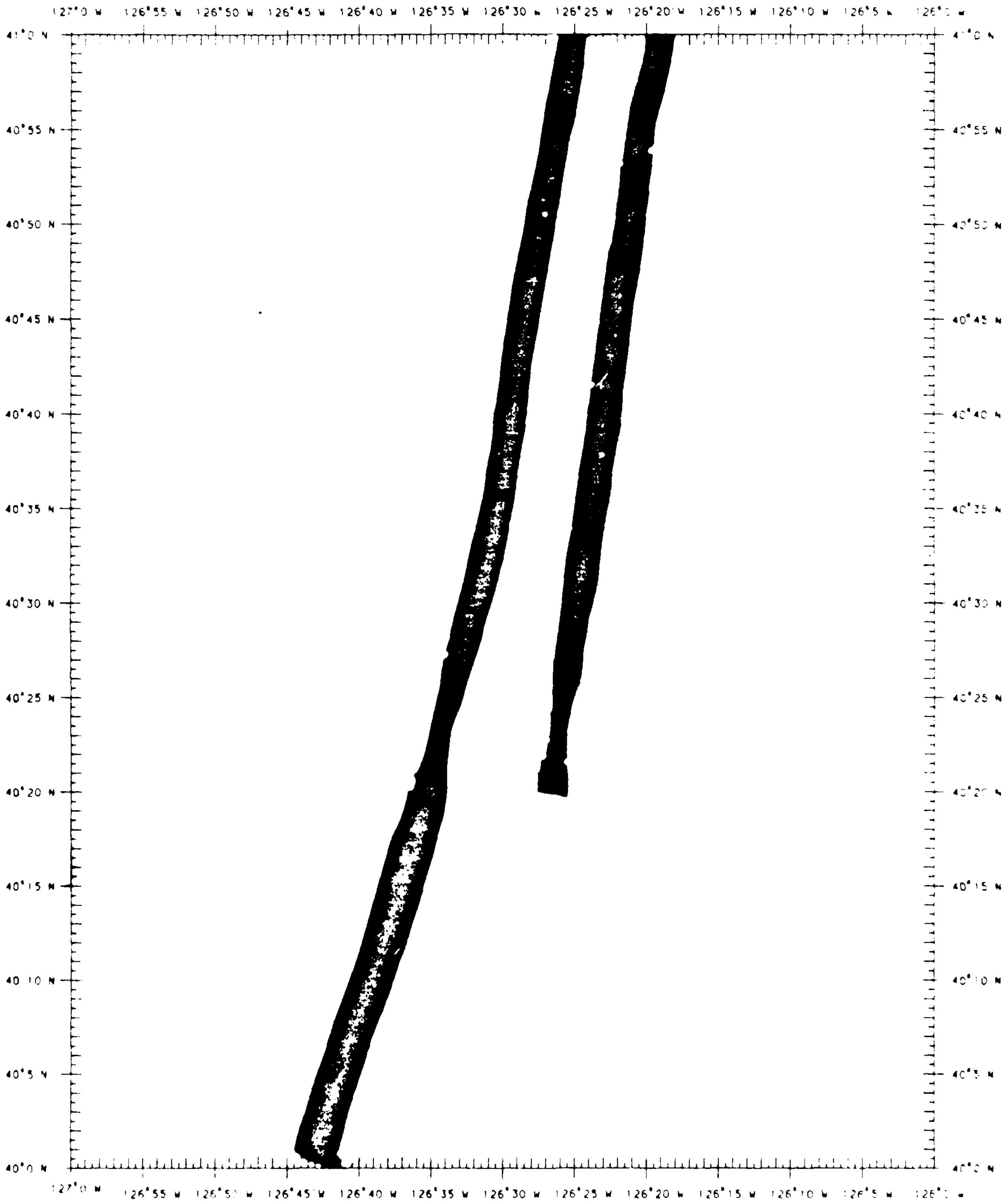
```
ebyt=(i >> 3) + buf;
sbit=str % 8;
ebit=i % 8;
i=ebyt-sbyt-1;
if(sbyt == ebyt){
        num = ((*sbyt >> (8-ebit)) & ~msk[8-n]);
        }
else{
        num = *sbyt++ & ~msk[sbit];
        while(i--){
                num <<= 8;
                num |= *sbyt++;
                }
        num <<= ebit;
        num |= *sbyt >> (8-ebit);
        }
return num;
}
```

Appendix E

Data Sets and Graphics

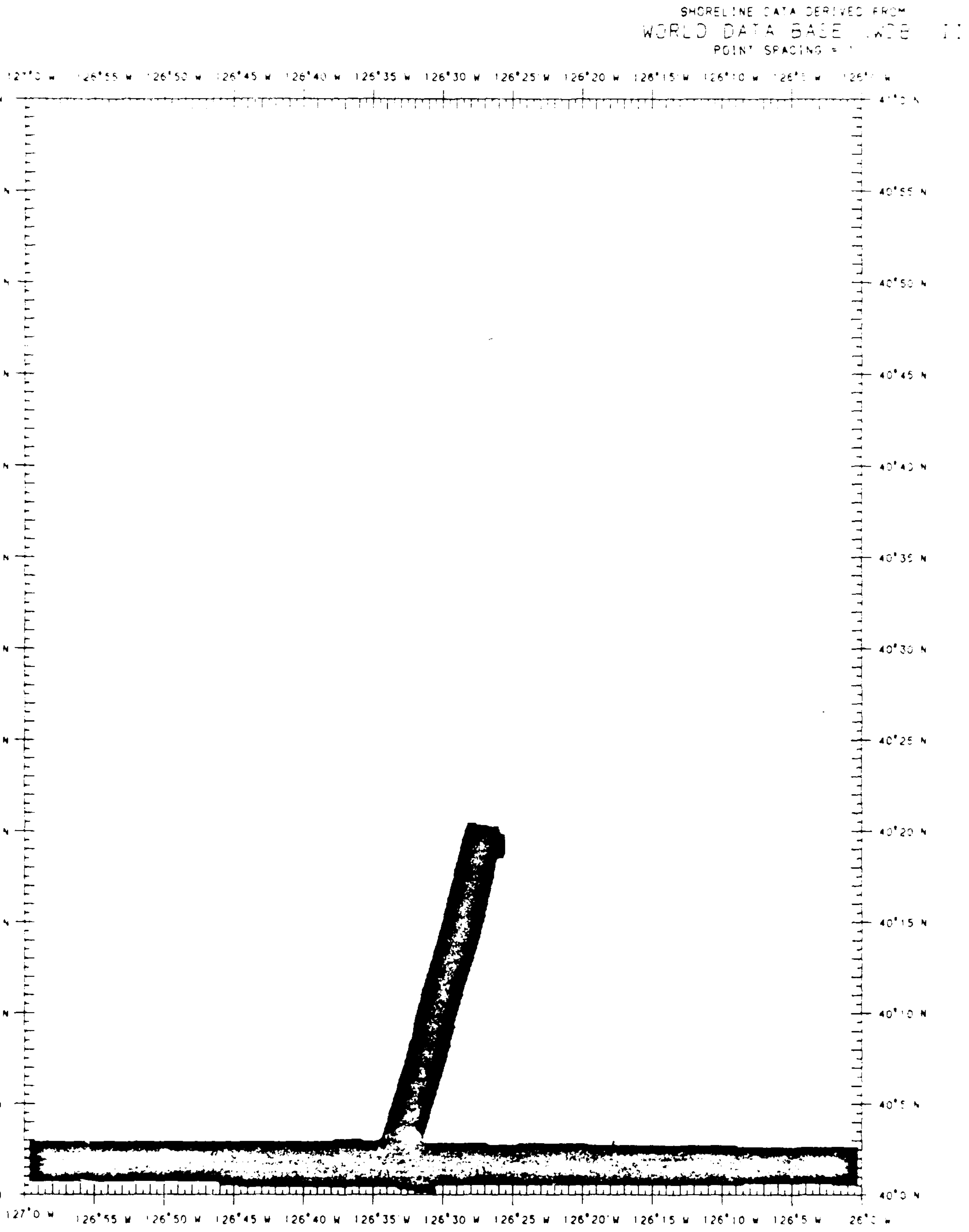
The following 12 graphics depict the swath coverage of each of the 12 data sets that were provided by the Naval Oceanographic Office. The original graphics were computer generated and provided on mylar sheets with each data set represented by a different color. Data within the shaded areas was provided in digital form with a 0.1 minute resolution.

SHORELINE DATA DERIVED FROM
WORLD DATA BASE (WDB) II
POINT SPACING = 1



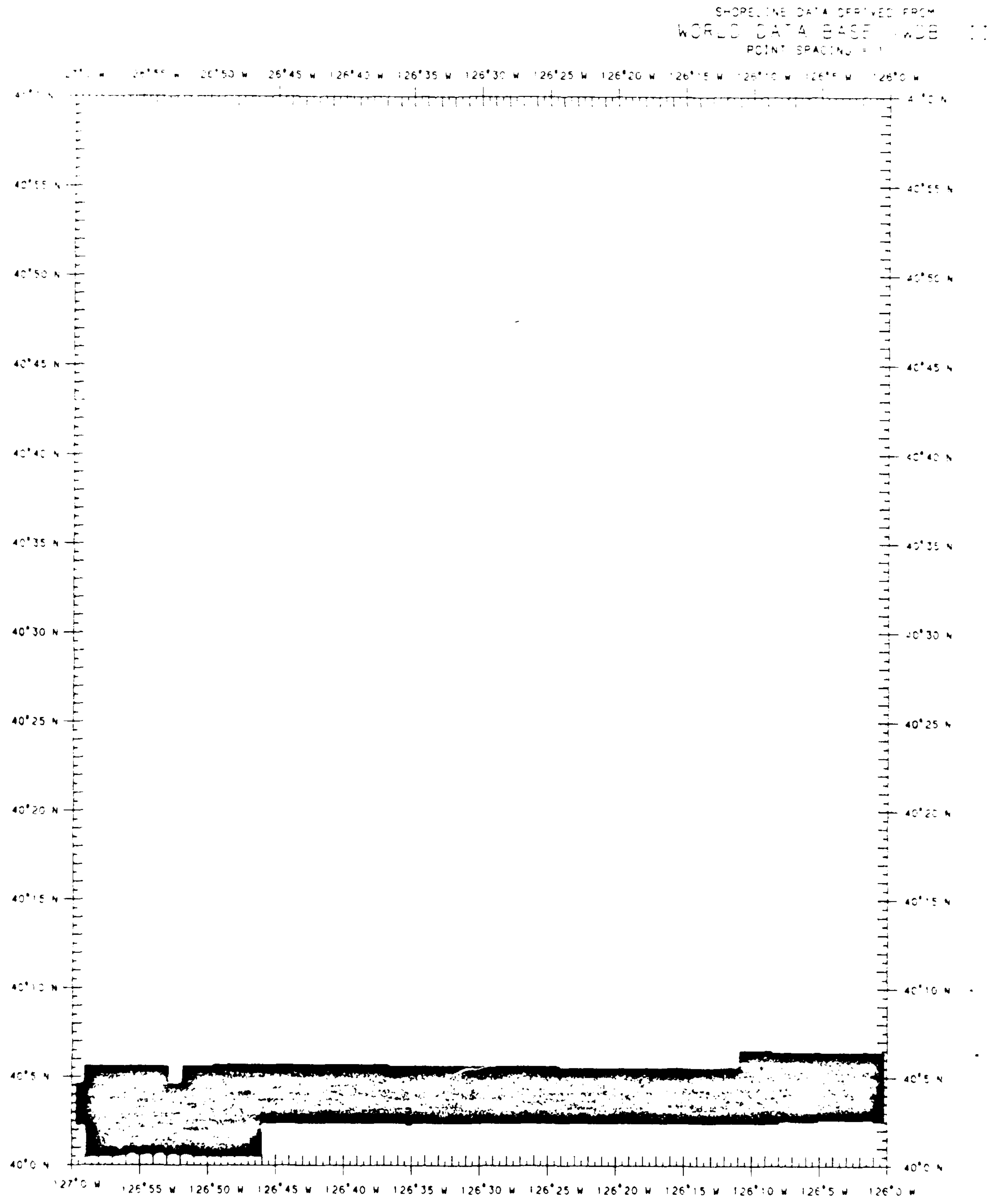
WGS84 MERC PRJ. PS:8. SPECIAL DATA BASE

DEPTHS IN FATHOMS



WGS84 MERC PROJ. PS:8, SPECIAL DATA BASE

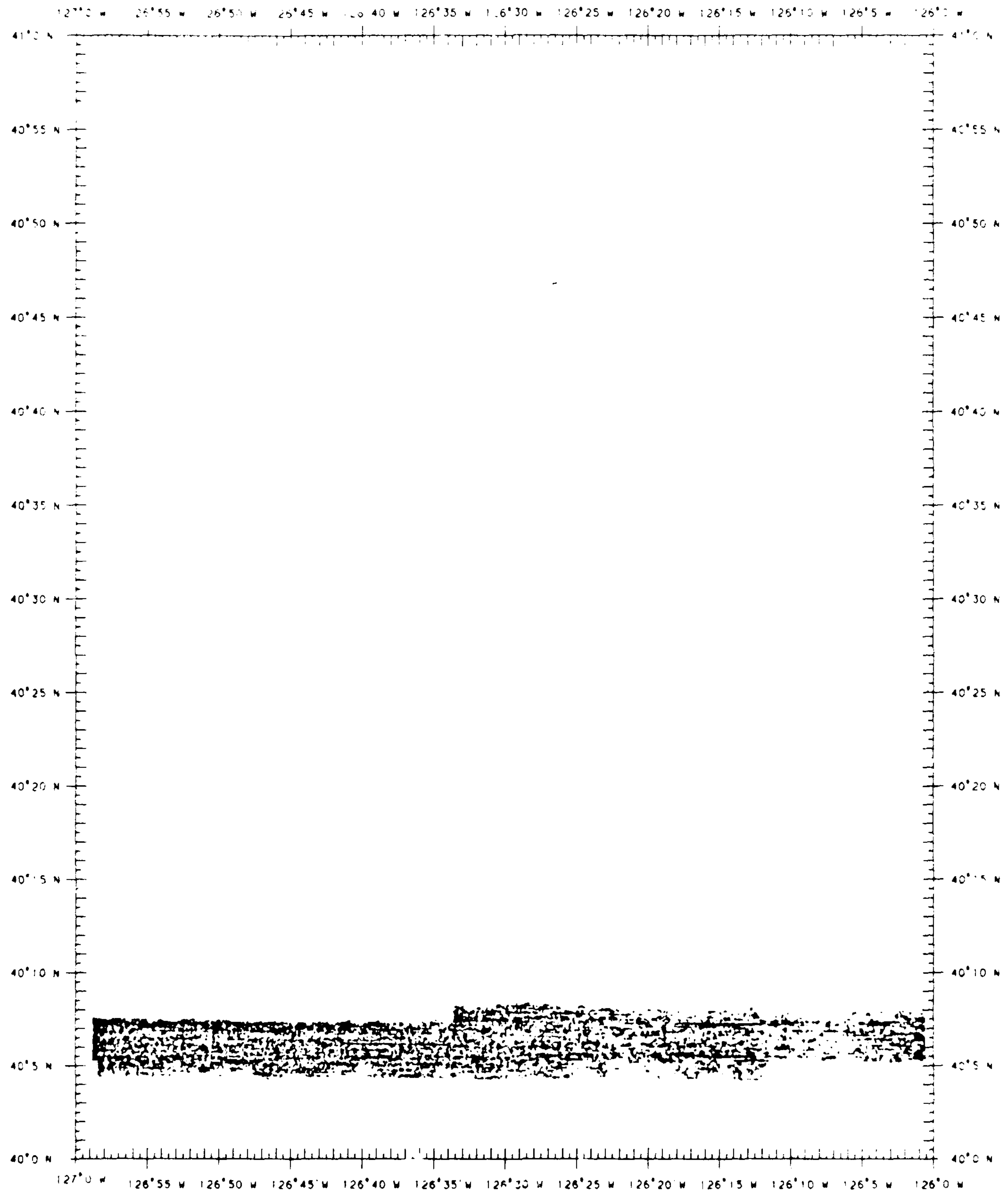
DEPTHS IN FATHOMS



WGS84 MERC PRJ. PS:8. SPECIAL DATA BASE

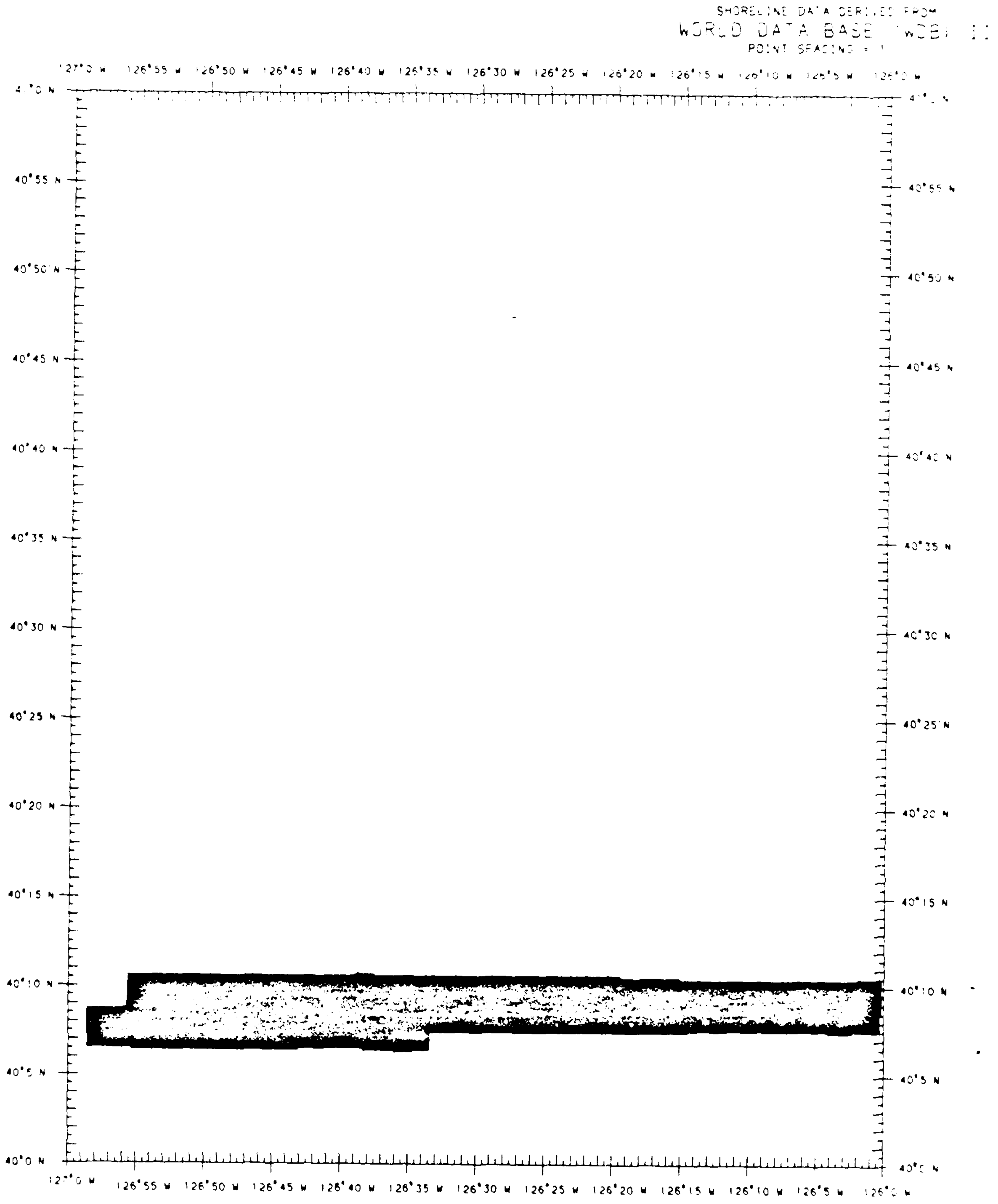
DEPTHS IN FATHOMS

SHORELINE DATA DERIVED FROM
WORLD DATA BASE (WDB II)
POINT SPACING = 1



WGS84 MERC PROJ. PS<8. SPECIAL DATA BASE GRID #1

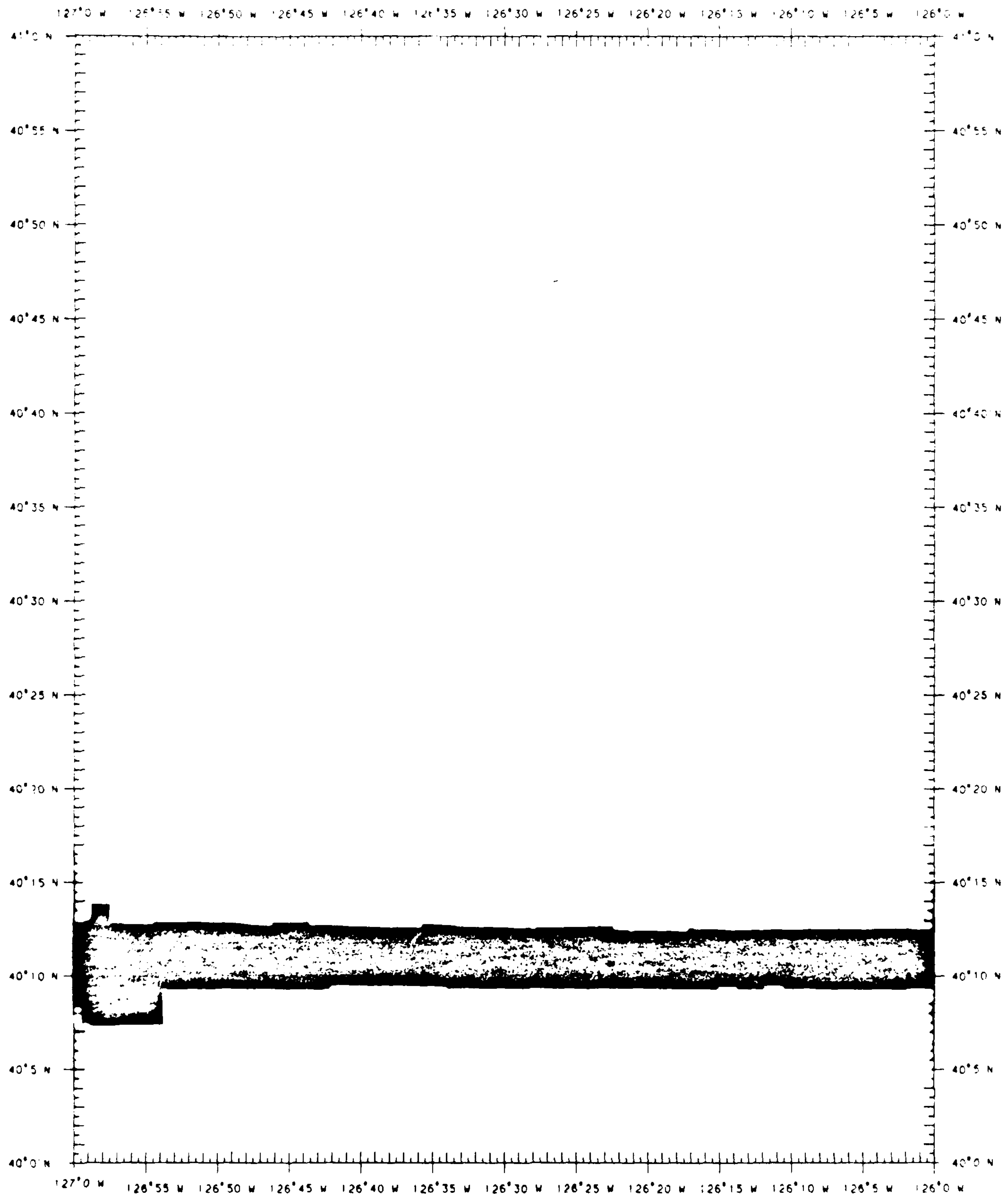
DEPTHS IN FATHOMS



WGS84 MERC PRJ. PS:8. SPECIAL DATA BASE

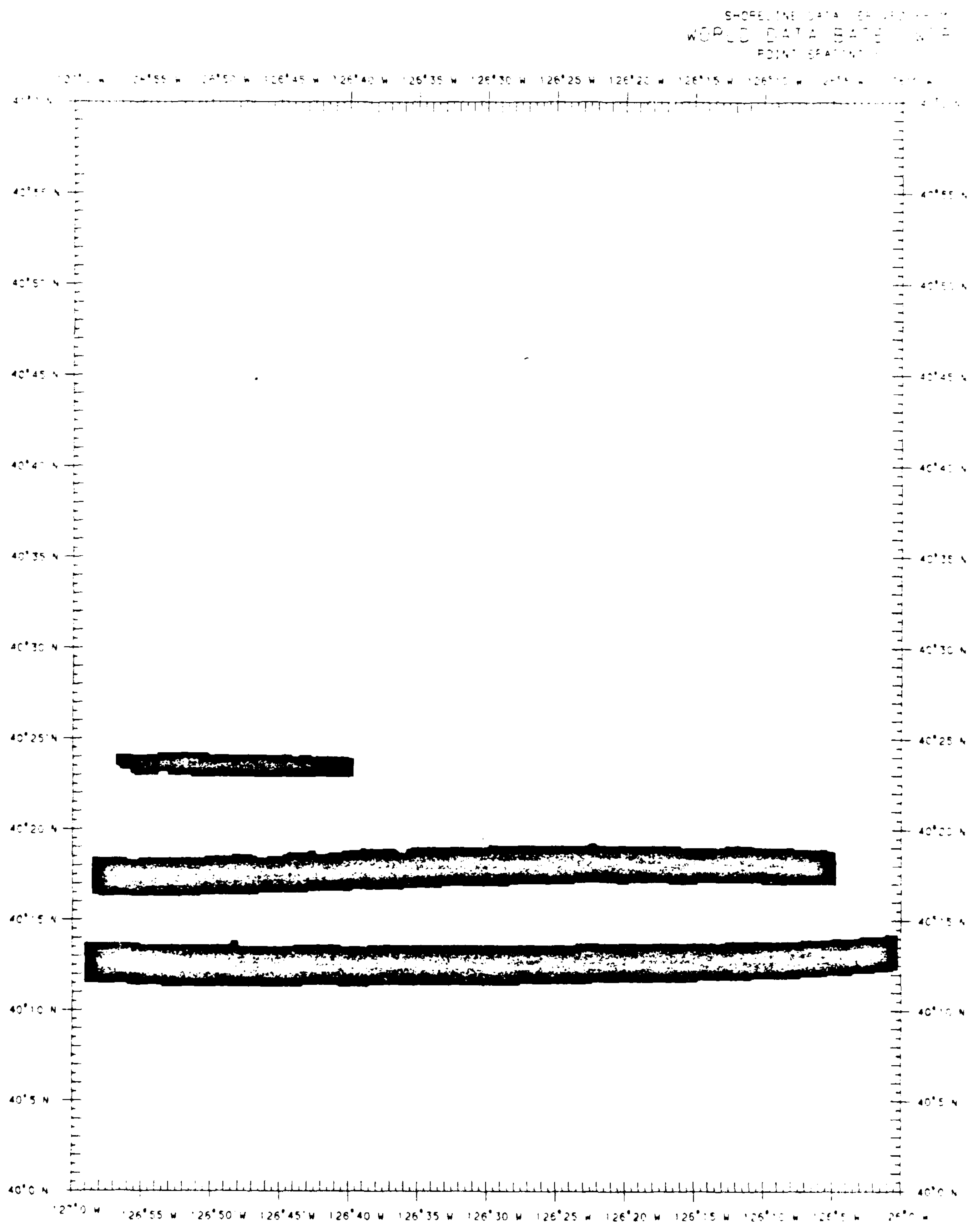
DEPTHS IN FATHOMS

SHORELINE DATA DERIVED FROM
WORLD DATA BASE (WDB II)
POINT SPACING = [illegible]



WGS84 MERC PRJ. PS=8. SPECIAL DATA BASE

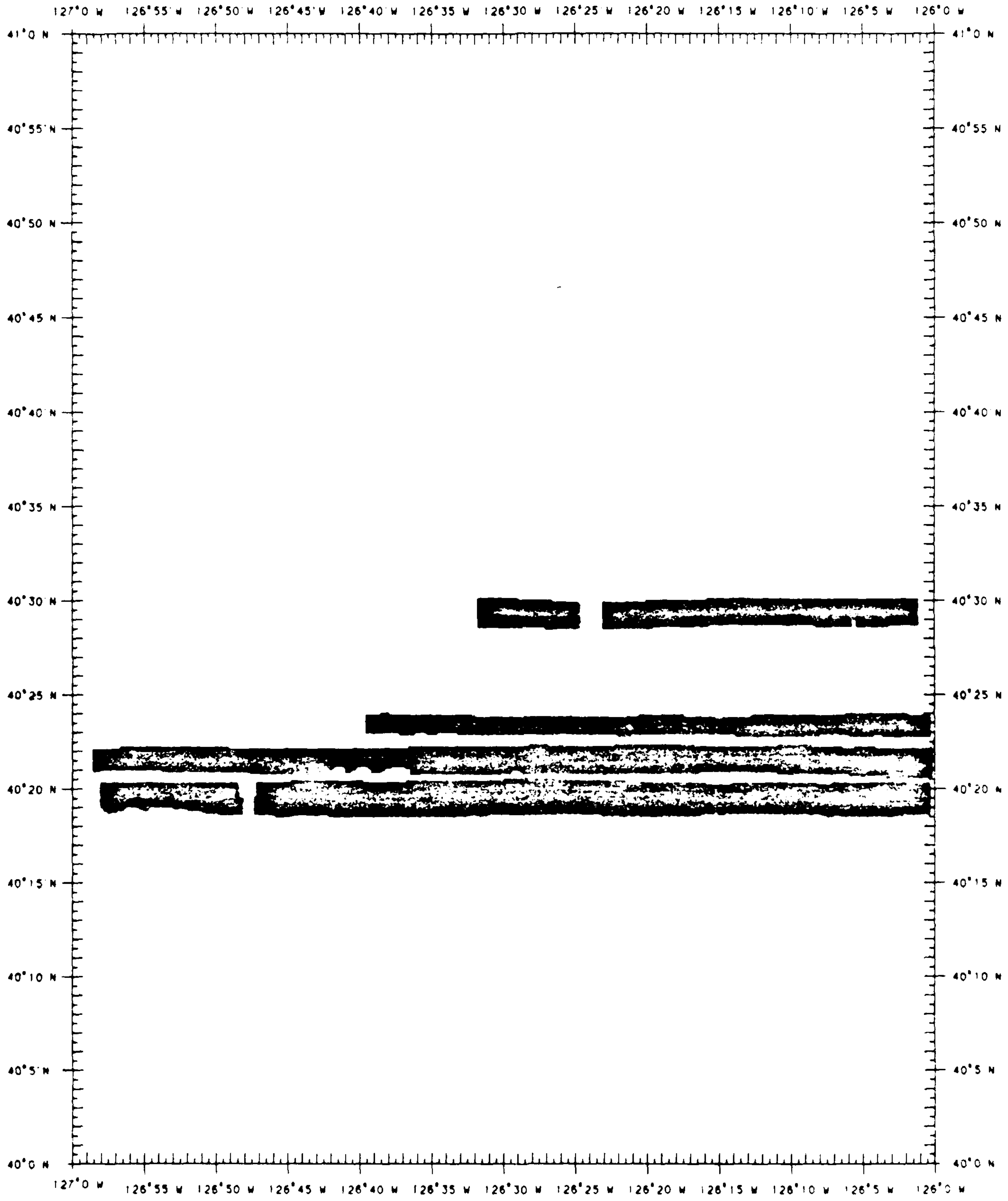
DEPTHS IN FATHOMS



WGS84 MERC PRJ. PS:8. SPECIAL DATA BASE

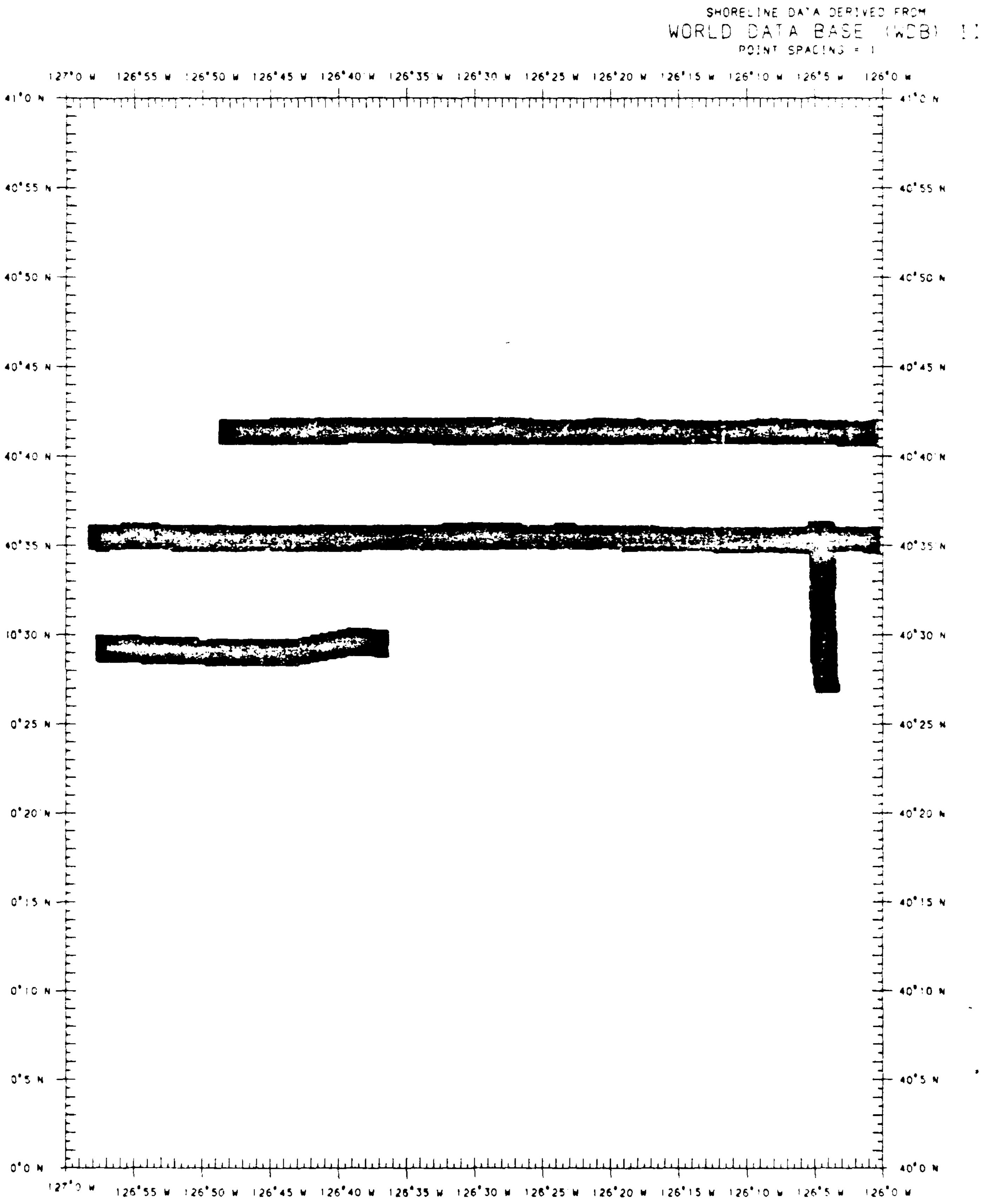
DEPTHS IN FATHOMS

SHORELINE DATA DERIVED FROM
WORLD DATA BASE (WDB) II
POINT SPACING = 1



WGS84 MERC PRJ. PS:8. SPECIAL DATA BASE

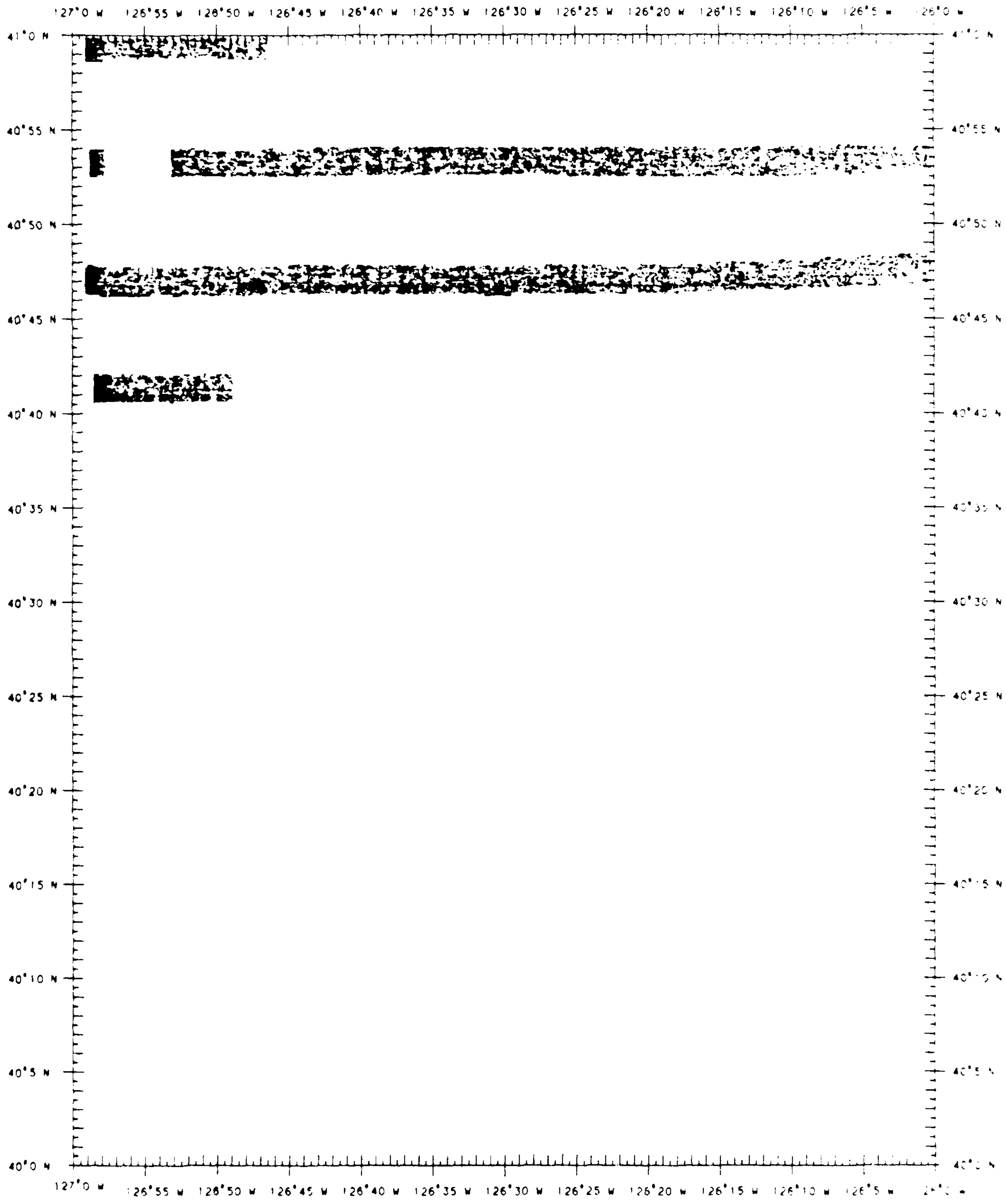
DEPTHS IN FATHOMS



WGS84 MERC PRJ. PS18. SPECIAL DATA BASE

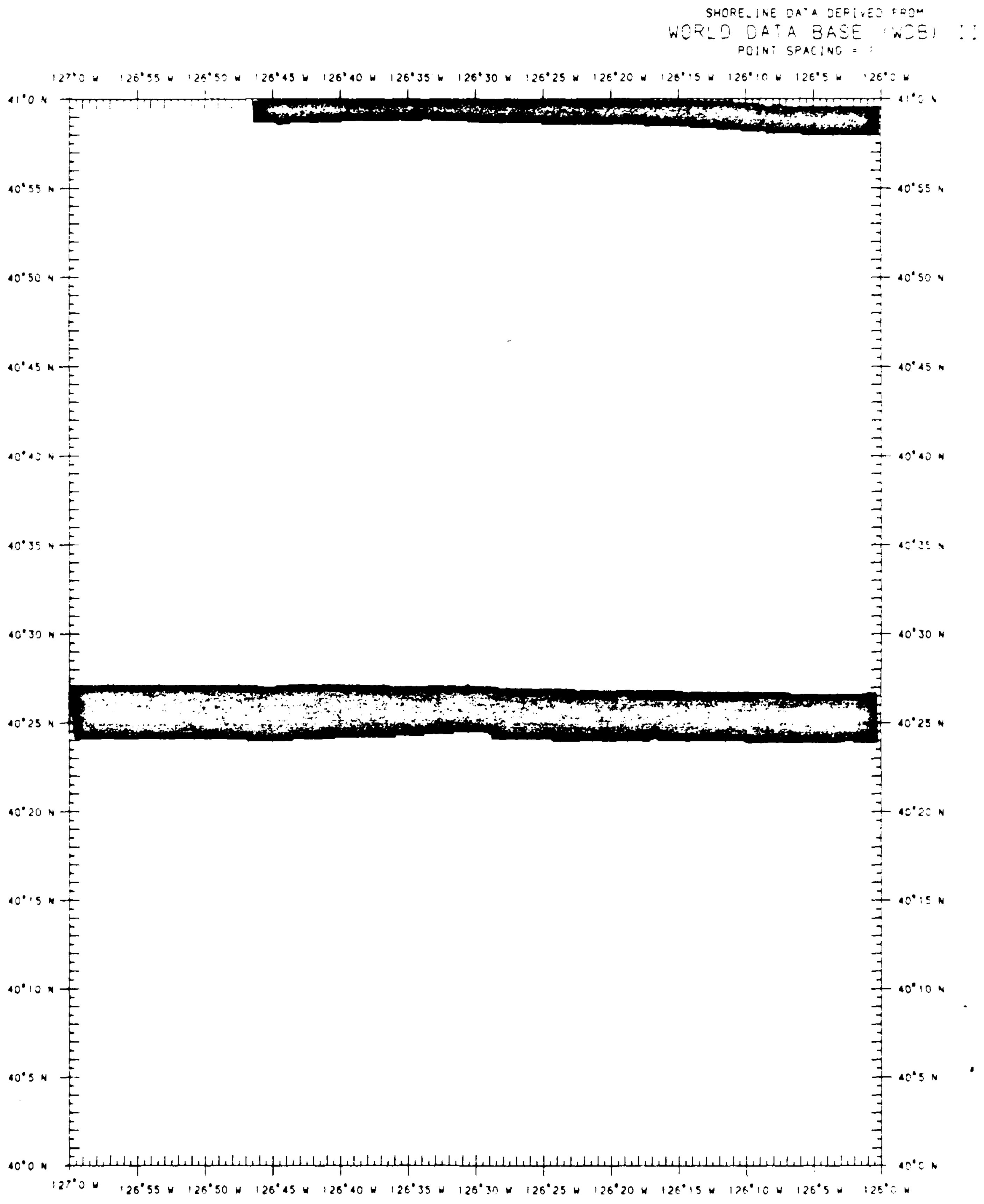
DEPTHS IN FATHOMS

SHORELINE DATA DERIVED FROM
WORLD DATA BASE (WDB II)
POINT SPACING = 1



WGS84 MERC PRJ. PS:8. SPECIAL DATA BASE

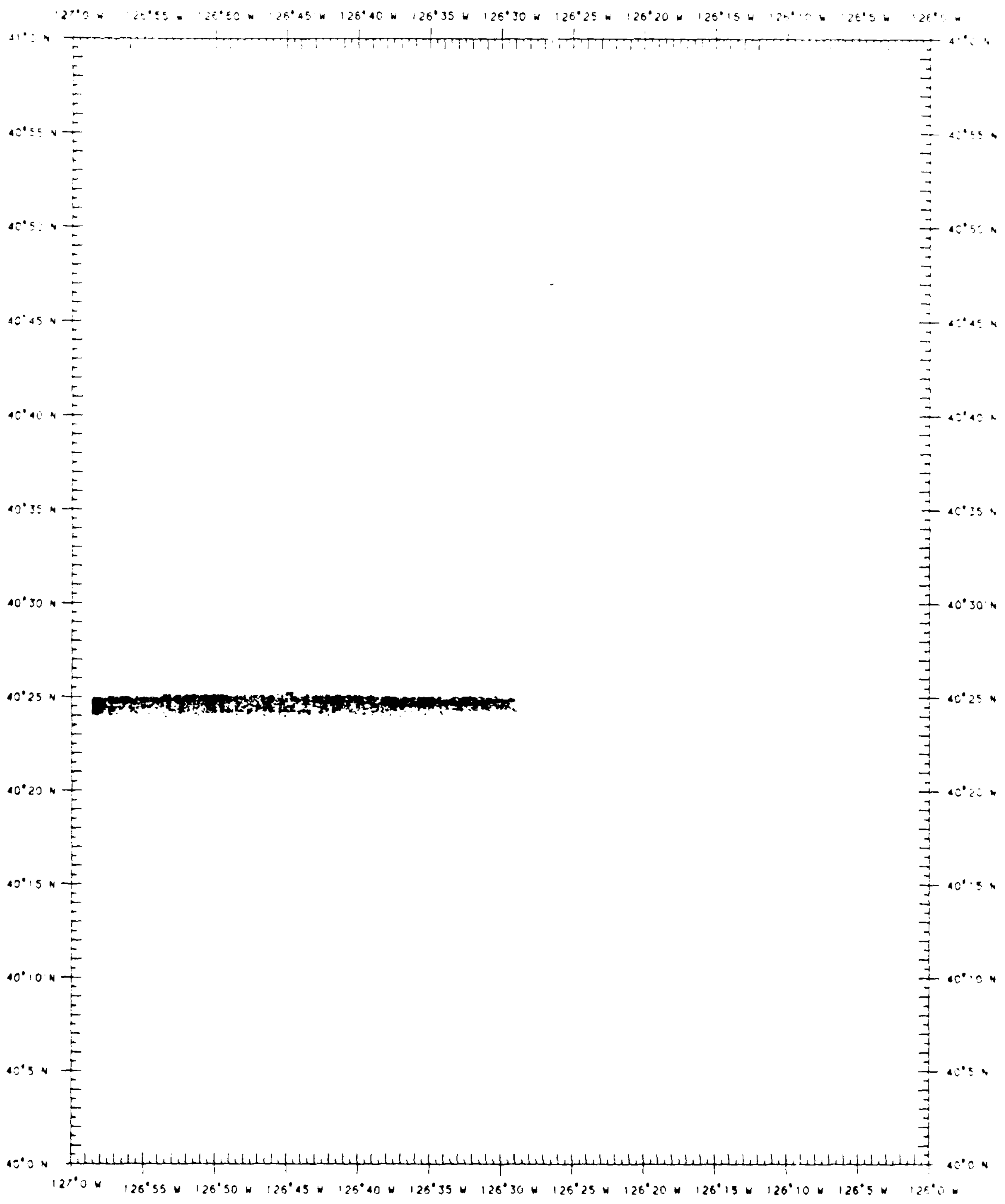
DEPTHS IN FATHOMS



WGS84 MERC PRJ. PS:8. SPECIAL DATA BASE

DEPTHS IN FATHOMS

SHORELINE DATA DERIVED FROM
WORLD DATA BASE WDB II
POINT SPACING = 1



WGS84 MERC PRJ. PS:8. SPECIAL DATA BASE

DEPTHS IN FATHOMS

Distribution List

Asst Secretary of the Navy
(Research, Engineering & Systems)
Navy Department
Washington DC 20350-1000

Chief of Naval Operations
Navy Department
Washington DC 20350-1000
Attn: OP-02
OP-71
OP-0962X, R. Feden
OP-987

Oceanographer of the Navy
Chief of Naval Operations
U.S. Naval Observatory
34th & Mass Ave., NW
Washington DC 20390-1800
Attn: OP-96

Commander
Naval Air Development Center
Warminster PA 18974-5000

Commanding Officer
Naval Coastal Systems Center
Panama City FL 32407-5000

Commander
Space & Naval Warfare Sys Com
Washington DC 20363-5100

Commanding Officer
Naval Environmental Prediction
Research Facility
Monterey CA 93943-5006

Commander
Naval Facilities Eng Command
Naval Facilities Eng Command HQ
200 Stovall St.
Alexandria VA 22332-2300

Commanding Officer
Naval Ocean R&D Activity
Stennis Space Center MS 39529-5004
Attn: Code 100
Code 105
Code 115
Code 117, J. Hammack
Code 125EX
Code 125L (13)
Code 125P (1)
Code 200
Code 300
Code 350, D. Hickman
Code 351, J. Byrnes
Code 351, J. Braud
Code 351, M. Lohrenz

Brooke Farquhar
NORDA Liaison Office
Crystal Plaza #5, Room 802
2211 Jefferson Davis Hwy.
Arlington VA 22202-5000

Commanding Officer
Naval Research Laboratory
Washington DC 20375

Commander
Naval Oceanography Command
Stennis Space Center MS 39529-5000

Commanding Officer
Fleet Numerical Oceanography Center
Monterey CA 93943-5005

Commanding Officer
Naval Oceanographic Office
Stennis Space Center MS 39522-5001
Attn: Code GGAP, B. Mullen
Code A, J. Depner
Code G, G. Caruthers
Code GB, O. Avery
Code GBA, D. Doyle

Commander
Naval Ocean Systems Center
San Diego CA 92152-5000

Commanding Officer
ONR Branch Office
Box 39
FPO New York NY 09510-0700

Commander
David W. Taylor Naval Research Center
Bethesda MD 20084-5000

Commander
Naval Surface Weapons Center
Dahlgren VA 22448-5000

Commanding Officer
Naval Underwater Systems Center
Newport RI 02841-5047

Superintendent
Naval Postgraduate School
Monterey CA 93943

Director of Navy Laboratories
Rm 1062, Crystal Plaza Bldg 5
Department of the Navy
Washington DC 20360

Officer in Charge
New London Laboratory
Naval Underwater Sys Cen Det
New London CT 06320

Director
National Ocean Data Center
WSC1 Room 103
6001 Executive Blvd.
Rockville MD 20852
Attn: G. W. Withee

Director
Woods Hole Oceanographic Inst
P.O. Box 32
Woods Hole MA 02543

University of California
Scripps Institute of Oceanography
P.O. Box 6049
San Diego CA 92106

Officer in Charge
Naval Surface Weapons Center Det
White Oak Laboratory
10901 New Hampshire Ave.
Silver Spring MD 20903-5000
Attn: Library

Commanding Officer
Fleet Anti-Sub Warfare Training Center,
Atlantic
Naval Station
Norfolk VA 23511-6495

Defense Mapping Agency Sys Cen
12100 Sunset Hill Rd. #200
Reston VA 22090-3207
Attn: SGWN
Code 10D/10P, Dr. E. Silva
Mel Wagner
Ed Danford

Office of Naval Technology
800 N. Quincy St.
Arlington VA 22217-5000
Attn: Code 20, Dr. P. Selwyn
Code 228, Dr. M. Briscoe
Code 234, Dr. C. V. Votaw

Office of Naval Research
800 N. Quincy St.
Arlington VA 22217-5000
Attn: Code 10
Code 10D/10P, Dr. E. Silva
Code 12
Code 112, Dr. E. Hartwig

Commander
Naval Sea Systems Command
Naval Sea Systems Command HQ
Washington DC 20362-5101

Commanding Officer
Naval Civil Engineering Laboratory
Port Hueneme CA 93043

Commander
Naval Air Systems Command
Naval Air Systems Command HQ
Washington DC 20361-0001

Pennsylvania State University
Applied Research Laboratory
P.O. Box 30
State College PA 16801

University of Texas at Austin
Applied Research Laboratories
P.O. Box 8029
Austin TX 78713-8029

Johns Hopkins University
Applied Physics Laboratory
Johns Hopkins Rd.
Laurel MD 20707

University of Washington
Applied Physics Laboratory
1013 Northeast 40th St.
Seattle WA 98105

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. Agency Use Only *(Leave blank)*

2. Report Date.
November 1989

3. Report Type and Dates Covered.
Final

4. Title and Subtitle.
Data Base Structure to Support the Production of the Digital Bathymetric Data Base

5. Funding Numbers.

Program Element No.	980101
Project No.	00101
Task No.	0102C
Accession No.	DN257116

6. Author(s).
James E. Braud, John L. Breckenridge, James E. Current, and Jerry L. Landrum

7. Performing Organization Name(s) and Address(es).
Ocean Science Directorate
Naval Ocean Research and Development Activity
Stennis Space Center, Mississippi 39529-5004

8. Performing Organization Report Number.
NORDA Report 236

9. Sponsoring/Monitoring Agency Name(s) and Address(es).
Naval Oceanographic Office
Stennis Space Center, Mississippi 39529

10. Sponsoring/Monitoring Agency Report Number.

11. Supplementary Notes.

12a. Distribution/Availability Statement.
Approved for public release; distribution is unlimited. Naval Ocean Research and Development Activity, Stennis Space Center, Mississippi 39529-5004.

12b. Distribution Code.

13. Abstract *(Maximum 200 words).*

The Naval Ocean Research and Development Activity was tasked and sponsored by the Naval Oceanographic Office to develop a computer model that would support the storage and retrieval of the 7 billion points of collected bathymetry that are used to support the Ocean Survey Program. These data points are stored at a resolution that is beyond the noise level induced by the collection sensors. Gridding these bathymetric data to an acceptable resolution of 0.1 minute for areas of actual coverage can greatly reduce storage requirements. Further reductions can be achieved through the use of the compression algorithms developed as a result of this project. An overall compression ratio of 500 to 1 was demonstrated using actual data supplied by the Naval Oceanographic Office.

Once the data base is reduced in size to a manageable level, relational data base management systems become practical for organizing information related to the bathymetry in order to provide easy multikeyed access. The Structured Query Language, a de facto standard for data base retrieval, can be used to query the data base to obtain tables of path names for the files that contain the compressed bathymetry. A suggested format for these files and the relational data base structure is presented in this report.

14. Subject Terms.
bathymetry, data bases, prototypes

15. Number of Pages.
52

16. Price Code.

17. Security Classification of Report.
Unclassified

18. Security Classification of This Page.
Unclassified

19. Security Classification of Abstract.
Unclassified

20. Limitation of Abstract.